

The Pennsylvania State University

The Graduate School

College of Engineering

**A Novel Approach of Successive Interference Cancellation
with an Alternative Physical Layer**

A Thesis in

Electrical Engineering

by

Alexis Sietins

© 2009 Alexis Sietins

Submitted in Partial Fulfillment
of the Requirements for
the Degree of

Master of Science

August 2009

The thesis of Alexis Sietins was reviewed and approved* by the following

Mohsen Kavehrad
Professor of Electrical Engineering

Randy Young
Graduate Faculty Member
Thesis Adviser

Ken Jenkins
Professor of Electrical Engineering
Head of the Department of Electrical Engineering

*Signatures are on file at the Graduate School

Abstract:

Communications drives the world. Much research is being done to advance the abilities to communicate with more people or higher data rates. Bandwidth has become the limiting factor for many communication systems. One popular solution for efficient bandwidth utilization is Direct Sequence Spread Spectrum (DSSS) which is used as the physical layer modulation underneath Code Division Multiple Accesses (CDMA).

In Direct Sequence – Code Division Multiple Access (DS-SS), every user acts as an interferer to every other user. This fundamental issue limits the number of users that can be placed on a single frequency band. The implementation of Power Control, causing the power of every user at the receiver to be nearly the same, increased the capacity of the channel. One newer technique being analyzed to improve DS-SS channel capacity is Multi-user Detection (MUD) or multi-user interference mitigation. One type of interference mitigation is Successive Interference Cancellation (SIC). Currently, SIC is not feasible to be implemented in mobile devices due to of the computation time and power consumption needed.

Scale-Time Offset Robust Modulation (STORM) is being presented in this paper as an alternative physical layer to help with synchronization for SIC on a DS-SS channel.

Table of Contents

List of Figures	v
List of Tables	vi
List of Abbreviations	vii
1 Introduction	1
1.1 Background	1
1.2 Multi-user Detection	1
1.3 Subtractive Interference Cancellation	2
2 Successive Interference Cancellation	3
2.1 Successive Interference Cancellation	3
2.2 New Approach	4
3 Scale-Time Offset Robust Modulation	5
3.1 STORM	5
3.2 STORM Modulation Description	6
3.3 Time-Offset/embedded Reference Modulation	6
3.4 STORM's Transmit Waveform Design	7
3.5 STORM Demodulation	8
3.6 STORM and SIC	10
4 Simulation	13
4.1 Setup	13
4.2 MATLAB Transmission Description	13
4.2.1 User ID's	14
4.2.2 Channel Coherence	14
4.2.3 Pulse Shaping	16
4.3 MATLAB SIC Description	16
4.3.1 Power ordering & Synchronization	17
4.3.2 Data Demodulation and Remodulation	18
4.4 Results	19
4.5 Conclusions	20
Bibliography	21
Appendix : Documentation of MATLAB files	25
1. coherent_detector.m	25
2. data_collection.m	28
3. decode_frame.m	29
4. demod_storm_pulse_shape.m	34
5. find_sync.m	40
6. generate_user_IDs.m	42
7. power_order.m	44
8. remod_signal.m	45
9. remove_frame_from_signal.m	48
10. transmit_storm_pulse_shape.m	50

List of Figures

Figure 1	Optimal MUD technique	2
Figure 2	Time Offset Modulation	7
Figure 3	STORM Modulator	8
Figure 4	STORM Modulation	9
Figure 5	STORM Correlation Receiver	10
Figure 6	graphical STORM demodulation	11
Figure 7	Flow Chart of SIC using STORM	12
Figure 8	Demonstration Setup	14
Figure 9	Channel Coherence for 44.1kHz and 40kHz	16
Figure 10	Signal Bandpass Filter Response	16
Figure 11	Pulse Shape Filter	17
Figure 12	3-d STORM Surface	18
Figure 13	2d STORM surface projection	19
Figure 14	Frame Correlation Pre and Post SIC	20
Figure 15	Frame Correlations of SIC Twice	21

List of Tables

1. Partial preferred polynomials for $N = 10$	1
---	-------	---

List of Abbreviations

BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CDMA	Code Division Multiple Access
DS-CDMA	Direct Sequence – Code Division Multiple Access
FDMA	Frequency Division Multiple Access
FSK	Frequency Shift Keying
MAI	Multiple Access Interference
MLBS	Maximum Length Binary Sequences
MLSE	Maximum Likelihood Sequence Estimation
MMSE	Minimum Mean Squared Error
MUD	Multi-User Detection
PN	Pseudo Random
QAM	Quadrature Amplitude Modulation
SIC	Successive Interference Cancellation
SS	Spread Spectrum
SSMA	Spread-Spectrum Multiple Access
STORM	Scale-Time Offset Robust Modulation
TDMA	Time Division Multiple Access
TR	Transmitted Reference

1 Introduction

1.1 Background

One wireless networking solution is CDMA, in where users share the same channel in time and frequency. Unlike frequency-division multiple access (FDMA), where users are separated by orthogonal frequency bands, and time-division multiple access (TDMA), where users are separated by orthogonal time slots, users in CDMA obtains multiplexing by use of unique codes. Since every user can transmit at any point in time and is also allocated the entire frequency band, CDMA is also known as a spread-spectrum multiple access (SSMA) method [1].

In DS-CDMA, the most popular version of CDMA, each user's data is multiplied by a unique code. The received signal is created by summing all users' signals. Each users' signal overlaps in time and frequency with every other user. Signal detection and demodulation, for a particular user, is traditionally performed by correlating to the unique code of the particular user [1].

Multiple access interference (MAI), the name given to the interference between direct sequence users, establishes a restriction on the capacity and performance of a DS-CDMA system. DS-CDMA is an asynchronous communications system. Therefore, there are timing offsets between signals that makes it impossible to design unique user code waveforms to be completely orthogonal resulting in MAI. MAI is a function of the number and the power of the users within the channel, as either value increases so does the MAI. The conventional detector ignores MAI in by using a single-user approach for signal detection [1].

Multi-user detection (MUD) is a better approach for signal detection. MUD is also known by the names joint detection and interference cancellation. MUD exploits information about more than one user simultaneously to better detect an individual user and their data. Use of MUD techniques has the potential to further aid current DS-CDMA systems [1].

1.2 Multi-user Detection

MUD exploits the fact that the structure of the MAI is known. With this in mind, the interference can be detected and removed. For this to be successful though, an accurate estimate of the MAI needs to be obtained. MUD can be divided into two broad subgroups: Optimal MUD and Sub-optimal MUD.

Optimal MUD uses either Maximum Likelihood Sequence Estimation (MLSE) or a Matched Filter Bank followed by the Viterbi Algorithm [26]. The Viterbi Algorithm samples the output from the match filters in sequential order and then calculates the most likely path solution. An example of the use of the Viterbi Algorithm method can be seen in Figure 1.

The problem with using Optimal MUD Solutions is that the complexity is not realistic in a practical scenario. MLSE has 2^M solutions to be tested, where M is the

number of users on the channel. Use of the Viterbi Algorithm reduces the number of solutions to $2^{(M-1)}$. However, $2^{(M-1)}$ is still not a reasonable number of computations in a realistic scenario [26].

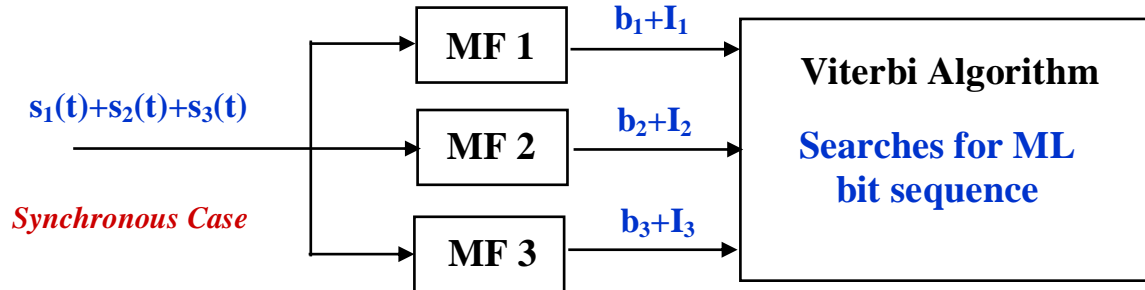


Figure 1 – Optimal MUD technique [26]

Sub-optimal schemes focus on reducing the complexity of the system to a more realistic level. This area can be divided into Linear and Non-Linear methods. Linear methods include Minimum Mean Squared Error (MMSE) and the Decorrelator. Non-Linear forms can come in Decision Feedback and Subtractive Interference Cancellation [26].

Using a Sub-optimal MUD scheme has several drawbacks in comparison to the optimal version. Non-Linear versions are susceptible to a common Spread Spectrum (SS) issues like “The Near-Far” problem. In comparison, linear receivers are less affected by this issue. Synchronizing to the received signal can also be an issue in non-linear schemes. Linear MUD techniques do not need to synchronize whereas Non-Linear receivers need to worry about when the signal starts. Good estimation of other channel characteristics are needed to be able to demodulate using a Non-Linear method [26].

1.3 Subtractive Interference Cancellation (SIC)

A particular group of non-optimal detectors that have received a lot of attention are known as successive interference cancellation (SIC) detectors. The driving concept of a SIC detector is to estimate the MAI created by each individual user and then remove a part or the whole contributed MAI from the aggregate received signal [1].

A hard or soft bit decision can be used to establish a particular users’ data [1]. The easier to implement solution is the soft-decision. Using a hard-decision bit estimate in a feed-back loop changes the SIC detector to be non-linear. An accurate estimate of the received power is needed for the hard-decision. However, hard-decision SIC detectors consistently perform better in comparison to a soft-decision SIC detector. Research has shown that when a users’ power estimate is imperfect, there is a degradation in the system performance [27,28].

2 Successive interference Cancellation

2.1 Successive Interference Cancellation

The SIC detector takes a serial approach to removing the MAI of a particular user [2, 3]. The detector first demodulates the signal to achieve bit estimates of the interfering signal. Next the detector regenerates an estimate of the interfering signal. Finally, the detector subtracts out the regenerated signal from the aggregate signal such that the remaining signal contains less MAI [1].

A simple pseudo algorithm of the SIC detector follows:

- Step 1. Power Order signals. Choose signal with the highest power.
- Step 2. Decode the data contained within the signal with the highest power
- Step 3. Remodulate the signal with the highest power.
- Step 4. Subtract out the interfering signal.

When step 3 provides an accurate estimate of the received interfering signal, this algorithm results in a correctly decoded data sequence for the interfering signal. Additionally, a new receive signal with lower MAI is produced.

This detector can be designed in an iterative method, where the k^{th} iteration takes in the input from the output of the $(k-1)^{\text{th}}$ iteration. The output of each subsequent iteration has a further reduction in MAI than the previous iteration. Additionally, the detector also outputs a bit estimate for the signal mitigated in the k^{th} iteration [1].

Choosing to mitigate the signals in descending order of powers is an easy choice. First, signal estimation and demodulation is easiest on users with the highest powers due to their correlation peaks being higher. Next, the mitigation of the users causing the most MAI will achieve the largest benefit for the resulting signal. In conclusion, mitigation of a user with lower MAI will not provide a significant benefit to a user with high MAI. However, the converse is not true. Mitigation of a user with high MAI will have a large benefit to a user with smaller MAI. However, this complicates the given algorithm further. First, an accurate estimate of each users power must be maintained at all times. Second, a further bit delay is introduced into the system. Thus, a compromise must be established between power ordering accuracy and computational complexity [1]. A problem occurs when the SIC detector incorrectly estimates a data sequence for a user. In this situation, irrelevant of the accuracy of the timing, power, and phase, subtracting out the regenerated signal will create more MAI than originally existed in the current iteration. In fact, instead of removing the MAI caused by the interfering user, the MAI is instead quadrupled in power. Consequently, a minimum bit error rate (BER) must be maintained within the system for the SIC detector to provide improvement in the system [1].

Many existing SIC implementations use conventional detectors to obtain and order the power estimations of the individual signals. By taking the correlation of every signal at each point in time, a correlation matrix can be formed. From this matrix, an

estimate of each signals power can be found and then reordered to continue the SIC process.

Two problems with using a conventional detector to create an estimate of the power are the resources required to create the correlation matrix, and the inflexibility of the correlation matrix in dealing with high energy multipaths. Creation of the correlation matrix at each point in time takes an abundance of resources and time. The extra resources needed for computation makes using a Computational Detector not a viable option for a mobile device. Strong powered multipaths cause a great problem because the joint correlation matrix becomes singular, and a singular matrix needs even more resources to be stabilized.

2.2 New Approach

A new method for implementing SIC has emerged. By using features of the physical layer implementation of Scale-Time Offset Robust Modulation (STORM) computationally cheaper and robust estimations of the power, phase, amplitude, and timing can be obtained.

3 Scale-Time Offset Robust Modulation

3.1 STORM

STORM [33, 40, 42, 43, 44] is a modulation scheme that enables highly flexible multi-resolution processing of signals through fading and dispersive channels. STORM may be used as a stand-alone modulation technique or it can be used as an encapsulation modulation technique containing a signal modulated with another modulation scheme, such as QAM [43]. STORM aids conventional systems that require rapid and robust synchronization. The benefits of using STORM are that it enables variable time resolution; rapid, robust, reliable signal detection and synchronization; unique spatial processing robustness and gains; estimates of signal power, controllable non-coherent multipath gain; and other, subsequently detailed benefits [44].

STORM is an extension of Transmitted Reference (TR) modulation in that it adds a time scaling step into the modulation process. Signal pairs in existing TR systems contain an offset in frequency or time but not both simultaneously. There are many limitations to a typical TR system. A TR system containing an offset in frequency requires additional bandwidth. Additionally, such a TR system is more difficult to synchronize to. There also exists a fundamental limitation on the frequency offset. When the channel coherence bandwidth is less than the frequency offset, there is a decorrelation in the frequency domain between the two signals and a subsequent degradation the system's ability to synchronize. A decorrelation between the signal pair also takes place in a time offset based TR system when the time delay surpasses the coherence time of the channel. A decorrelation in the time delayed TR system also has similar system synchronization performance degradation [43].

Similar to a TR signal, a STORM transmit signal is also obtained by summing a pair of relatively offset signals, known as the base and offset signals. The base signal can be comprised of many other modulation schemes or modulated signals (ex. random noise, colored noise, a PN sequence, or QAM). The offset signal is obtained from a copy of the base signal that is that modified by a time delay and time scale. Additionally, a phase or amplitude difference can also be applied to the offset signal. By using a time scale close to one, the base and offset signals will be highly overlapped in frequency. A similar overlap in time will occur when a time delay that is lower than the channel coherence time is used [43].

There is nothing to distinguish a TR time delayed offset signal from a multipath signal in a traditional channel. High levels of relative motion create natural time scaling distortion within electromagnetic signals of around $1 + 10^{-5}$ at velocities of 3km/s. Thus, significant time scaling does not occur naturally and therefore a time scaled signal can easily be distinguished from a multipath [42].

STORM waveform design offers several advantages. To begin with, the signal can rapidly and robustly detected and synchronized to even when the channel has pore time or frequency coherence. Additionally, by fixing the bandwidth, the multipath resolution is adjustable by varying the scaling value. Consequently, a STORM demodulator can stack

multipath energy from a non-coherent channel without using a RAKE receiver. Finally, processing rates for synchronization and detection are orders of magnitude lower than a traditional matched filter receiver, assuming the scale parameter is known in advance [42].

3.2 STORM Modulation Description

STORM is similar to time offset modulation, which is a TR modulation with an offset in time. Time offset modulation has been used in RF and fiber optic applications. The background of time offset modulation will be reviewed and will be expanded to the application of STORM [42].

3.3 Time-Offset/Embedded Reference Modulation

When the time scaling parameter of STORM has the value of one, the STORM modulation reduces to time offset modulation, which is a TR modulation with a time delay. Time offset modulation is created by the addition between a base signal and a modified copy of the base signal [42].

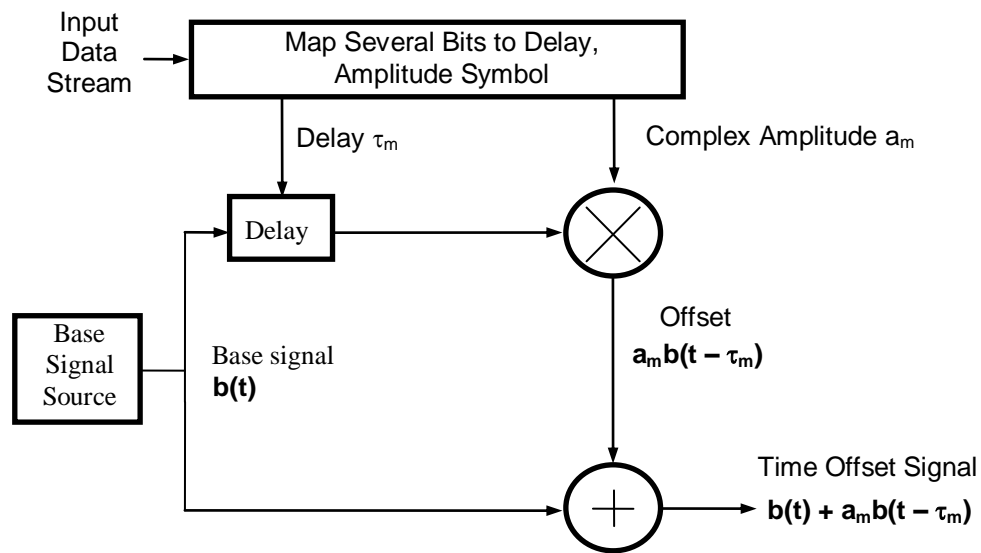


Figure 2 - Time Offset Modulation [42]

The base signal can be one of the following: wideband short duration, narrow band long duration, or wideband long duration. Possible specific signals or modulations include noise, PN sequences, BPSK, FSK, and QAM signals. Narrow band short duration signals are not used due to typically having ambiguous correlation features [43].

Figure 2 shows a schematic of the modulator for time offset modulation. To begin with, the base signal is obtained from a source. Next, a time delay is applied to a copy of the

base signal. To continue the modulation, the time delayed signal has a complex amplitude applied to it. Finally, the base signal is summed to the time delayed complex signal. The transmit signal is given by equation 1, where the base signal is represented by $b(t)$, the complex amplitude is a_m , and time delay is τ_m [42].

$$x(t) = b(t) + a_m b(t - \tau_m) \quad (1)$$

Equation 2, a common operator that estimates the auto-correlation of a signal over a period T , is used for specific lags in a time offset demodulator. When the delay in the demodulator corresponds to the delay in the modulator, a peak in the estimation of the auto-correlation will occur [42].

$$\hat{\phi}_{xx}(t_o, \tau) = \frac{1}{T} \int_{t_o}^{t_o+T} x(t)x^*(t-\tau)dt \quad (2)$$

3.4 STORM's Transmit Waveform Design

STORM is an extension of existing TR modulation by adding time scaling. Figure 3 shows that an input data stream establishes the information for delay, time scale, and amplitude. The delay term represents a time shift. Time scaling interpolates or decimates the signal in time. A phase and amplitude difference between the base and offset signals is established by multiplication of a complex amplitude [42].

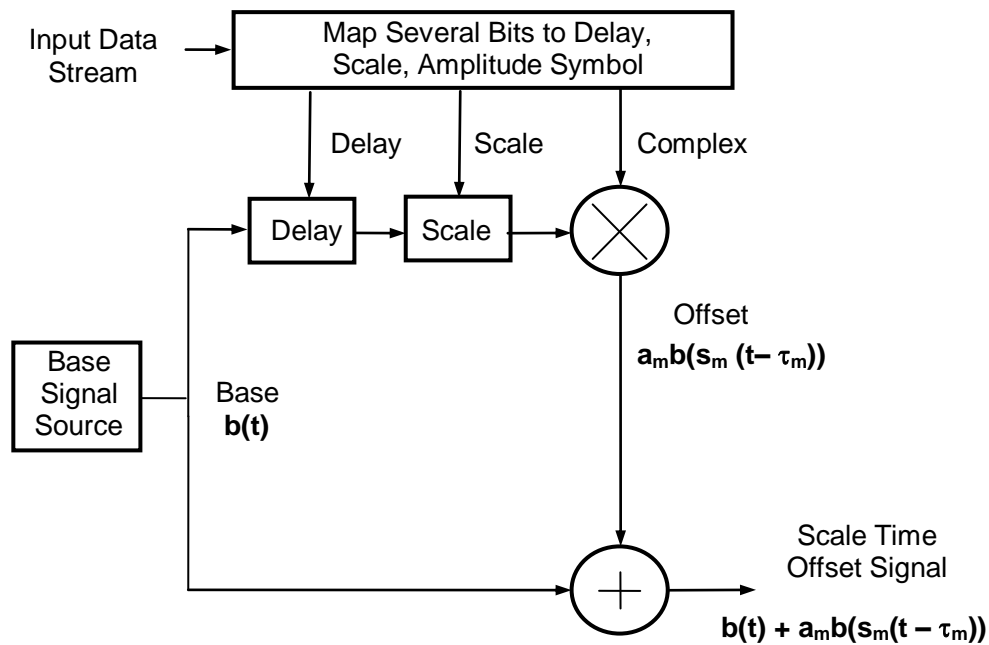


Figure 3 - STORM Modulator [42]

The four distinct steps in creation of a STORM signal are shown in Figure 4. First, a copy of the base signal is delayed in time by τ_m . Next, the copied and delayed signal is time

scaled by s_m . Thirdly, the time-scaled, delayed, and copied signal had a complex amplitude, a_m , applied to it. Finally, the offset signal is summed to the base signal. Equation 3 shows a non-orthogonal mathematical form of the STORM signal. The base signal is given by $b(t)$. The complex amplitude, time scale, and delay values are represented by a_m , s_m , and τ_m respectively.

$$x(t) = b(t) + a_m b(s_m(t - \tau_m)) \quad (3)$$

The offset signal can also be established by swapping the time scaling and time delay steps. This changes the equations, but the signal qualities do not change [42].

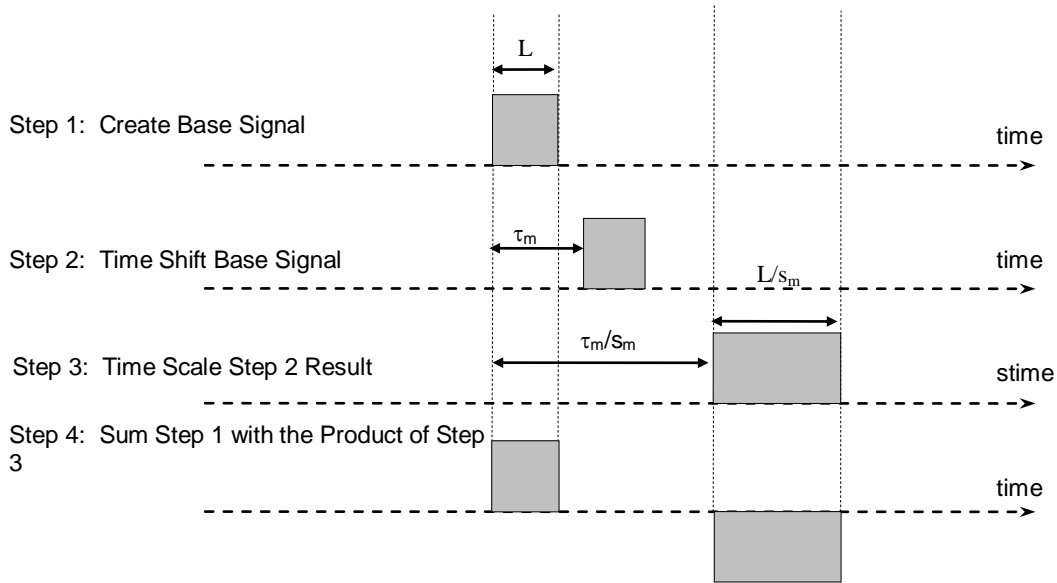


Figure 4 - STORM Modulation [42]

3.5 STORM Demodulation

One way to demodulate STORM is to apply the time scale to the received base signal so that it can be correlated to the received offset signal. Alternately, time scaling the received offset signal such that it will correlate with the received base signal would also work. Figure 5 shows a figure representing the first type of STORM demodulator. This is achieved by time delaying and time scaling a copy of the received signal to create an offset signal and then correlating the offset signal with the original received signal. For a high correlation peak to exist, s_i , the timing scale of the receiver, must match s_m , the timing scale of the transmitter [42].

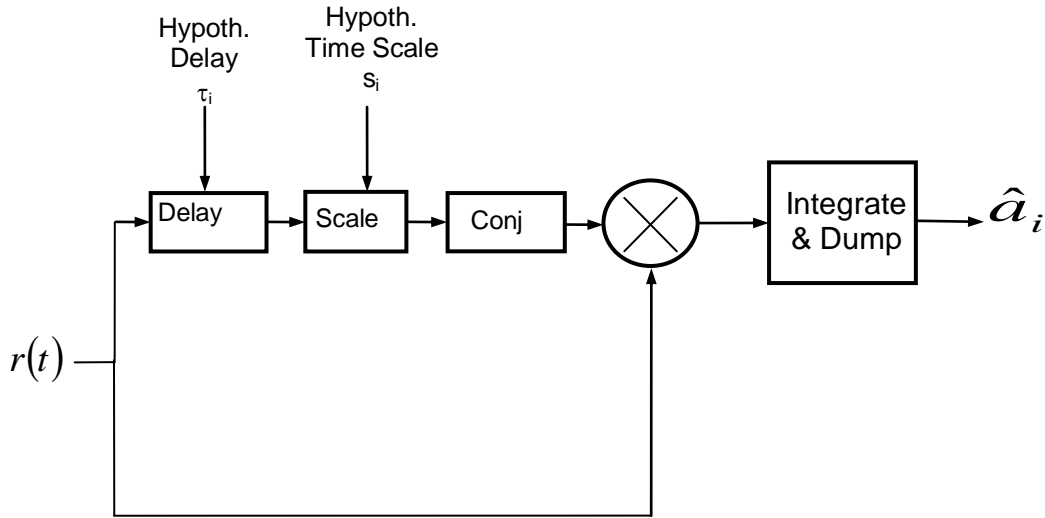


Figure 5 - STORM Correlation Receiver [42]

Equation 4 [42] defines a wideband auto-ambiguity estimator [5] for $x(t)$ from Figure 5:

$$\hat{\phi}_{xx}(t_o, s_i, \tau_i) = \frac{1}{T} \int_{t_o}^{t_o+T} x(t)x^*(s_i(t - \tau_i))dt \quad (4)$$

When the time scale, s_i , is equal to one, equation 4 reduces to the previously mentioned auto-correlation estimator. The value, t_o , represents the start of the hypothesized symbol period. The time delay and scale values are represented by s_i and τ_i , respectively. Estimating equation 4 one or more times may result approximations for a_m , s_m , and τ_m [42].

Figure 6 represents a pictorial look at a STORM demodulator. Equation 4, the wideband auto-ambiguity function, is estimated when the received signal, $r(t)$ is correlated with a time delayed and time scaled copy of $r(t)$. This system is known as an offset auto-correlator and is a function of the time delay and time scale [43].

A matched filter receiver must hypothesize the arrival of a signal with a minimum rate of the inverse of the bandwidth, $(1/BW)$. In other words, at a minimum sample rate for a signal, the matched filter receiver needs to do a correlation sample-by-sample. However, the STORM receiver does not need make a hypothesis at the same rate. For example, a STORM receiver might need to calculate 4-10 estimations of the auto-ambiguity function to achieve synchronization of a million sample signal. Few correlations are needed to achieve synchronization due to the fact that the base and offset signals maintain relative correlation across the entirety of a signal [43].

STORM Receiver: Offset Auto-correlation

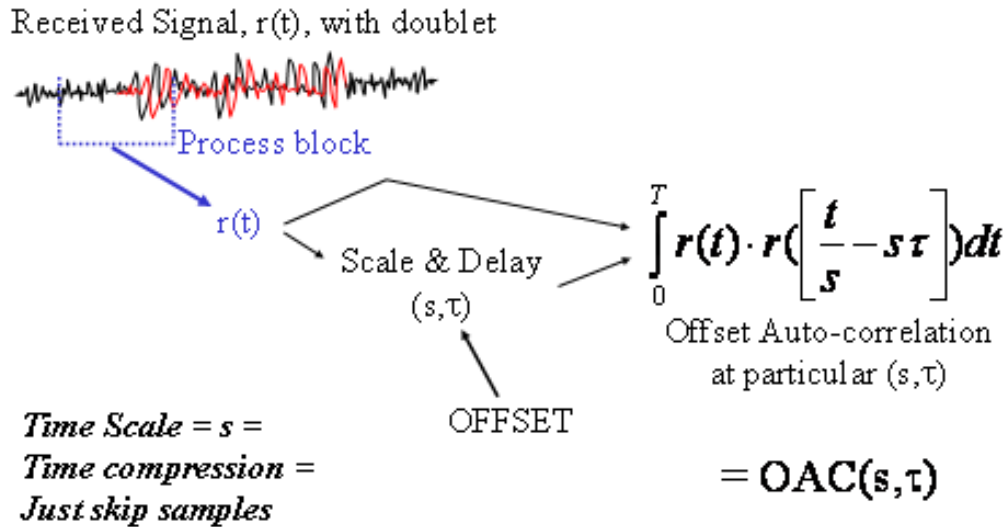


Figure 6 - graphical STORM demodulation [43]

3.6 STORM and SIC

The use of STORM as a physical layer aids in the implementation of SIC. STORM allows rapid and robust synchronization to a signal with fewer correlations than a traditional detector allowing for rapid power ordering of users and their multipaths.

Figure 10 shows a flowchart of SIC being implemented with the aid of STORM. This flow chart follows the SIC algorithm mentioned in chapter 2. Step 1 through step 3 of the STORM SIC algorithm match up with step 1 from the chapter 2 algorithm. T Blocks that have a MATLAB m-filename within the block are controlled by a MATLAB file of the labeled name. All MATLAB files are compiled within the Appendix. A breakdown of each block and functionality follows. A more detailed look into each block will follow in chapter 4. Step 1 through Step 3 of the STORM SIC method match up with step 1 of the pseudo-code SIC algorithm discussed in chapter 2.

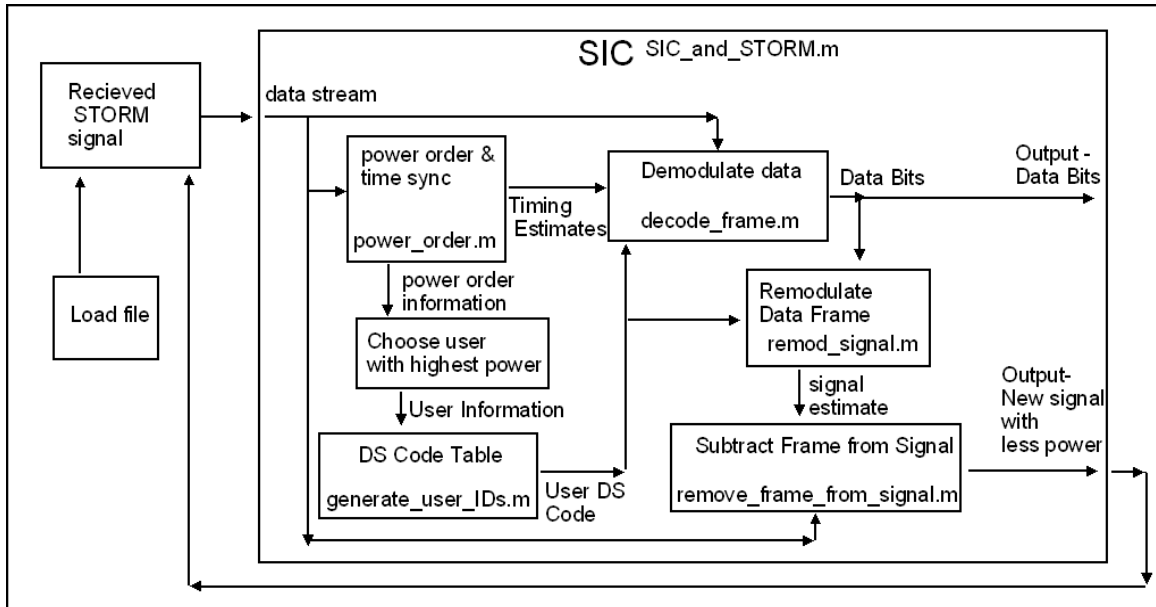


Figure 7 - Flow Chart of SIC using STORM

Step 1. Power Order & Rough timing estimate

The STORM synchronization process provides a power estimate of each user and each multipath. Additionally, a rough timing estimate is also provided.

Step 2. Sync to the signal with the highest Power

A hard decision is made choosing the signal with the highest power estimate.

Step 3. DS Code Table Lookup

A reference lookup for the direct sequence (DS) sequence for the chosen user takes place.

Step 4. Demodulate data

STORM's rough timing estimate for each user is not accurate enough for conventional demodulation, thus a standard detector follows the STORM demodulator. These methods are already well researched and documented [4]. The conventional detector will create bit estimates for the entire data frame.

Step 5. Remodulate

Step 1 provided a power and timing estimate of the signal. Step 4 provided estimate of the data bits for the given frame. Using these values, an estimation of the transmitted signal is created, $\hat{s}(t)$. The remodulation of the signal needs to accompany

both STORM and the accompanying modulation scheme. Thus, in the case of DS-CDMA, the remodulation step will need to account for the STORM and DS-CDMA aspects of the signal.

Step 6. Subtract

Once the estimate $\hat{s}(t)$ has been obtained, it is subtracted from the original received signal. The new signal, $s^*(t)$, will be returned to the beginning of the loop to be operated on again with the goal of having less Signal-to-Noise ratio than before.

4 Simulation

4.1 Setup

It has been argued in this paper that using STORM as a modulation technique provides a performance enhancement in the synchronization of signals over a conventional detector. A demonstration using MATLAB has been developed as a proof to this concept.

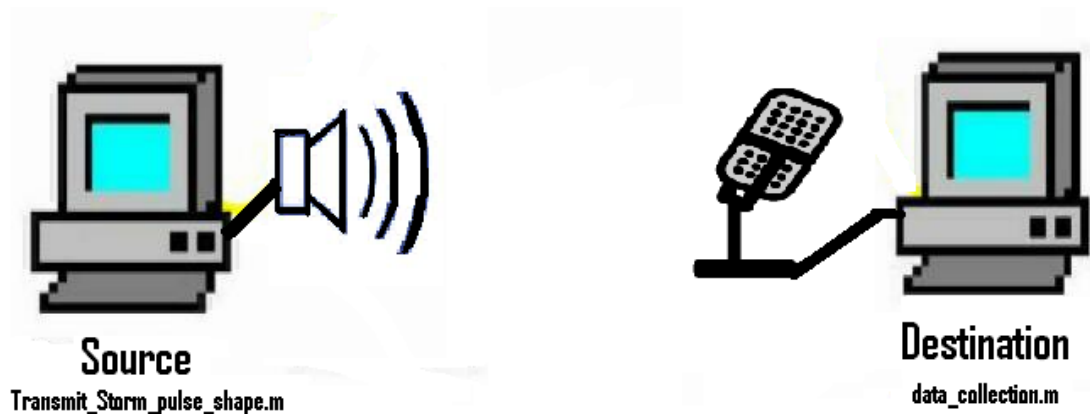


Figure 8 – Demo Setup

Figure 11 shows the setup for this demonstration. The source computer creates a baseband acoustic STORM signal which is transmitted using an attached speaker. The receiver computer records the acoustic signal via an attached microphone. The receiver computer then stores the signal on the hard drive. A second stage of the demonstration will load the saved data. The first stage of the demonstration, the transmitter and receiver setup, is discussed in 4.2. The second stage of the demonstration is discussed in section 4.3. The MATLAB files associated with transmitting and receiving are given below the respective stages in figure 11.

4.2 MATLAB Transmission Description

The STORM modulator, discussed in section 3.4 of this paper, was used to create the transmitter. The base signal source was chosen to be maximal length gold codes discussed in 4.2.1. Time scales were chosen to be less than one and have an order in the hundredths (ex .99, .98, and .97).

The base signal was created by modulating a random sequence of binary data bits with the user's DS code. The chip gain, the number of chips representing a single bit, was set to 100. A chip is a single sample of the DS and multiple chips are used to represent an individual bit.

Once the complete base signal was obtained, the base signal was copied and delayed by s_m . The delayed copy was then time-scaled by τ_m . The two signals, the base signal and the time-delayed and time-scaled signal, are summed together. The final composite signal was then passed through a pulse shaping filter followed by a bandpass filter to minimize inter-symbol-interference and to accommodate the channel coherence respectively. The bandpass filter is discussed in 4.2.2 and the pulse shaping filter is covered in section 4.2.3. The signal returned from the filters was then played through the computers speaker.

The receiver, using a microphone, records for a certain amount of time. The data recorded saved locally to a file on the hard drive.

4.2.1 User ID's

Maximal length Gold codes were selected as the DS user codes for the demonstration. A Maximal length binary sequence (MLBS) is known as a pseudo-random binary sequence. A MLBS is a periodic, deterministic, sequence with white noise properties.

A Gold code set comes in the size of 2^N , which is greater than the $2^{N/2}$ size of Kasami code set. However, Kasami codes have better cross-correlation performance than Gold codes. Where N is the number of shift registers used to create the sequences.

To generate a Gold code set, begin with two maximum length sequences, of length $2^N - 1$, $a[n]$ and $b[n]$. The two sequences, $a[n]$ and $b[n]$ must be limited to a three-values cross-correlation. Sequences that share only a three valued cross-correlation are known as preferred polynomials. A partial table for $N = 10$ is shown in Table 1 [40]. The modulo two addition of the two maximal length sequences with all phase shifts, including a shift of zero, comprises the 2^N Gold code set [40].

Primitive Polynomial 1	Primitive Polynomial 2
$1 + z^3 + z^{10}$	$1 + z^1 + z^2 + z^3 + z^5 + z^6 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^1 + z^3 + z^7 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^1 + z^2 + z^5 + z^6 + z^7 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^2 + z^3 + z^8 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^1 + z^5 + z^8 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^2 + z^5 + z^6 + z^7 + z^8 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^3 + z^4 + z^5 + z^6 + z^9 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^2 + z^3 + z^6 + z^8 + z^9 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^2 + z^4 + z^6 + z^8 + z^9 + z^{10}$
$1 + z^3 + z^{10}$	$1 + z^1 + z^3 + z^4 + z^6 + z^7 + z^8 + z^9 + z^{10}$

Table 1 – Partial preferred polynomials for $N = 10$

4.2.2 Channel Coherence

The STORM transmitter is limited to a sample rate of either 44.1kHz or 22.05kHz. This is due to the creation of an incoherent channel for other sampling rates chosen with MATLAB. Figure 12 shows a comparison of the channel coherence between a 44.1kHz

sample rate and a 40kHz sample rate. The 40kHz channel is not acceptable for use with a coherent demodulator, such as a matched filter. However, STORM can still synchronize to a signal in a non-coherent channel, which has been verified through simulation.

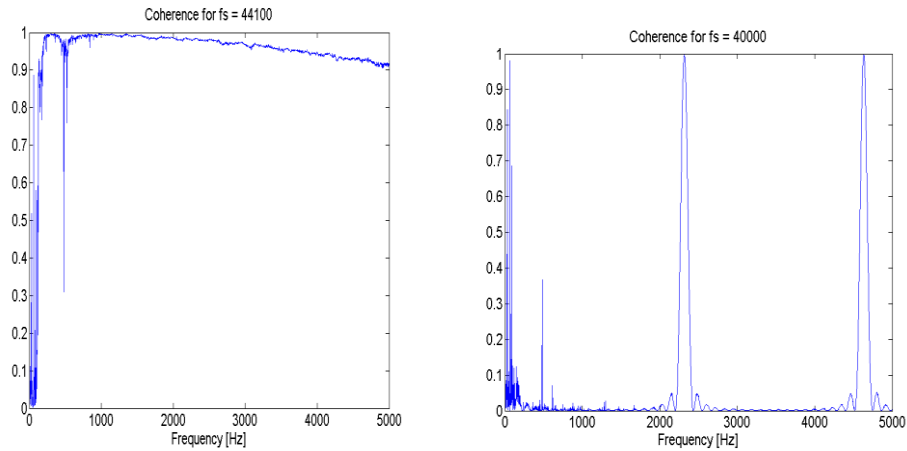


Figure 9 – Channel Coherence for 44.1kHz and 40kHz

A bandpass filter was chosen to be applied to the signal before it was transmitted. The 44.1kHz channel has poor channel coherence at less than 100Hz and starts to degrade past 3000Hz. The frequency response of the passband of the filter, shown in Figure 13, matches up with the coherent frequencies of the channel.

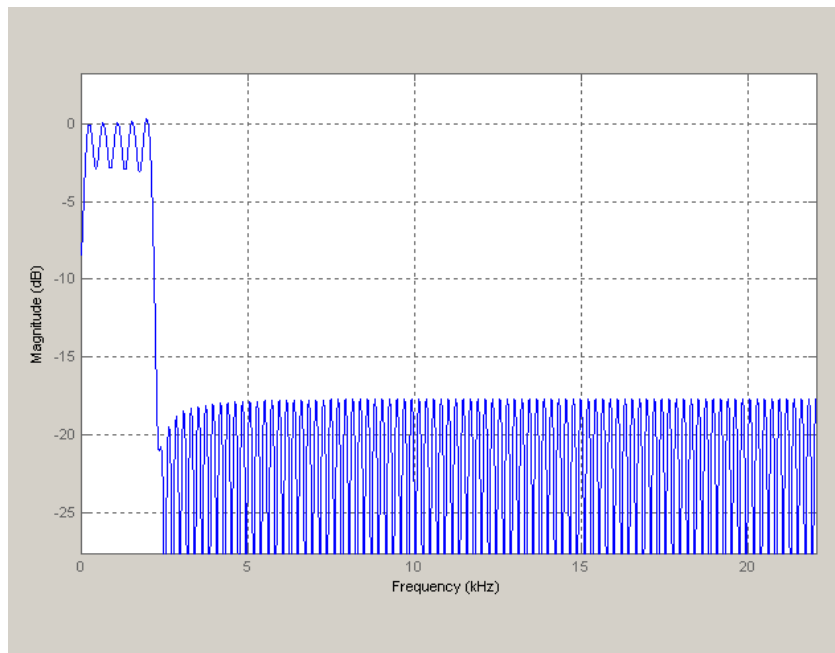


Figure 10 – Signal Bandpass Filter Response

4.2.3 Pulse shaping

To minimize inter symbol interference, the signal was pulse shaped by a root-raised-cosine filter. There is an upsample effect that occurs when applying a root raised cosine pulse shape to a digital signal. The upsample rate was chosen to be an even value of 10. The pulse shape filter used can be seen in figure 14.

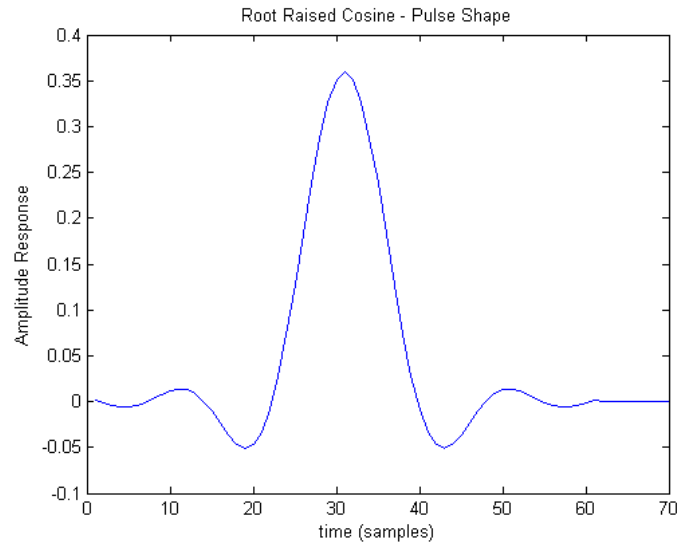


Figure 11 – Pulse Shape Filter

4.3 MATLAB SIC Description

The second stage of the demonstration, the state that applies SIC, follows the flow diagram given by figure 10. The first and second tiers of data flow are given by figure 10. The MATLAB file, SIC_and_STORM.m, represents the first tier view of the entire SIC system and is an encapsulation file that controls the entire second stage of the demonstration. This file follows the steps previously discussed in section 3.6 of this paper.

1. Power Order & Rough timing estimate
2. Sync to the signal with the highest Power
3. DS Code Table Lookup
4. Demodulate data
5. Remodulate
6. Subtract

A more detailed look at the Steps: 1 - Power Order, 2 - Synchronize, 4 – Demodulate, and 5 – Remodulate, will follow. Step 3 is a reference table lookup and will not be examined further. Power ordering and synchronization will be covered by section 4.3.1. Demodulation and remodulation will be discussed by section 4.3.2.

4.3.1 Power Ordering & Synchronization

An accurate estimate of each user's power is needed. The MATLAB file, `power_order.m`, provides these power estimates in addition to rough timing estimates. To obtain these needed values, the simulation goes through STORM demodulation as explained in Figure 8. The output of the STORM demodulator is a three-dimensional surface. Each received frame, whether a main signal or a multipath, shows up on this surface as different series of peaks.

Figure 15 shows one such STORM surface containing a single user's received data across two data frames. The correlation spreading effect of the STORM demodulator is also demonstrated within the figure. Instead of having a sharp impulse like peak, the output of the correlator has a spreading effect across time. The x and y axis in this figure represent time, in correlation shifts, and delay, in samples, respectively. The time shifts used to create the STORM surfaces within the demonstration are at .001 times the samples rate. Thus, there is a reduction computation for synchronization of a factor of 1000.

The peak of a particular STORM surface represents a power estimate of the scaled and delayed part of the STORM signal corresponding to the signal that created the STORM surface. For a power estimate of the composite STORM signal, the power estimate needs to be adjusted to include the base signal. Multiplying the power estimate by 2 results in an estimate with enough accuracy for the purposes of this demonstration.

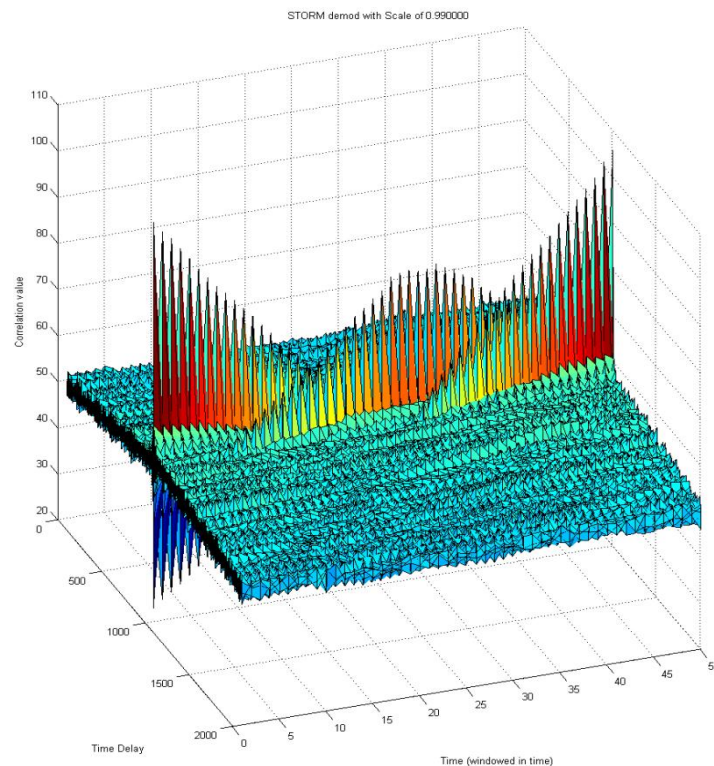


Figure 12 – 3-d STORM Surface

Following up the power estimate of the STORM surface is the synchronization step. To get the STORM synchronization, the line equation corresponding to the STORM surface must be calculated. The STORM surface is projected onto the time-delay correlation-index axis, creating a 2d representation of the STORM surface. This 2d representation can be seen in Figure 16. The x and y axis, like the STORM surface in Figure 15, are the correlation index and the time-delay values respectively. The line equation associated with the points along the STORM surface line can be calculated by using a second order linear least squares form fitting function. Methods for solving least squared functions are covered in the text [39]. The known time-delay, τ , is used within the line equation, $\tau = a_0 + a_1x$, and reverse solved to obtain the timing estimate. The units of the timing estimate are based off of the correlation index, and must be converted back into samples.

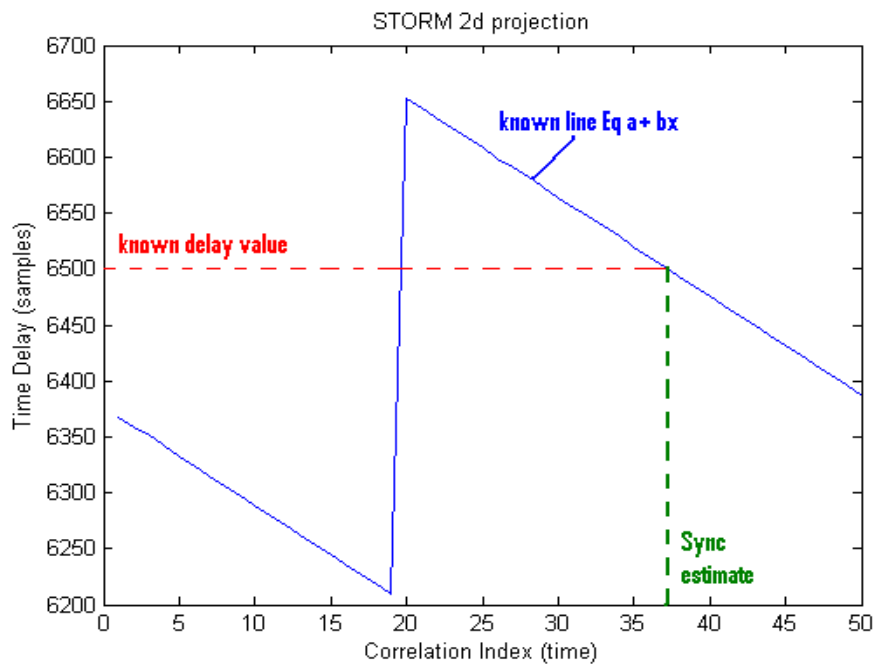


Figure 13 – 2d STORM surface projection

4.3.2 Data Demodulation and Remodulation

A traditional matched filter was chosen to demodulate the data and obtain the bit sequence. However, the matched filter was adjusted to take advantage of the fact that both the base signal and the time-delayed and scaled signal contain the transmitted data. I.e. the equation for the matched filter was equation 3 as opposed to the traditional $b(t)$.

Timing knowledge was exploited on the receiver for demodulation. The distance between each bit, in samples, in a frame was measured. The distance was assumed to be near constant for the entire frame, and any peaks from the correlator outside of the desired distance are assumed to be error. Following this idea, bit errors within the demonstration were reduced significantly.

It was found during simulation that the matched filter using equation 3 had a high sensitivity to τ_m and s_m . This means that if there is an error in the known values of τ_m and s_m , there will be a significant increase in the bit error rate.

The decoded bit sequence is then used to remodulate an estimate of the received signal. The power estimate established previously is then used to attenuate the remodulated signal to the desired power level.

4.4 Results

Making use of the ability to control whether the channel is coherent or not, simulations were ran to see if STORM could still synchronize within a non-coherent channel. It was found that even within a non-coherent channel, STORM was still able to produce an accurate power estimate and timing estimate. However, the simulator used a coherent detector, and therefore the data of the signal could not be decoded even though synchronization had occurred.

It was verified that STORM could synchronize to a signal with fewer correlations compared to a traditional matched filter. Consistent and successful synchronization occurred within the demonstration for a signal with a sample rate of 44.1 kHz and a correlation rate of 44 Hz. This is a reduction by a factor of a thousand in the needed correlations for synchronization compared to a traditional detector. It was also verified that STORM gave accurate power ordering of users at the same time as synchronizing to the respective users.

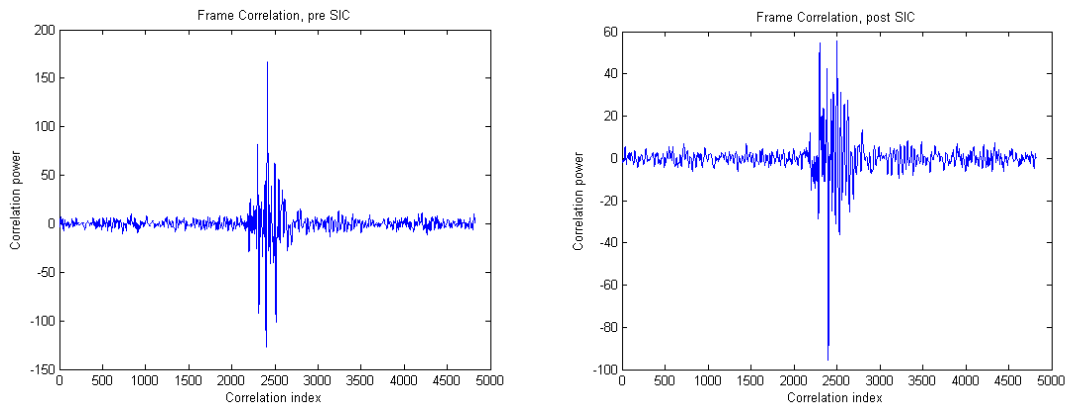


Figure 14 Frame Correlation Pro and Post SIC

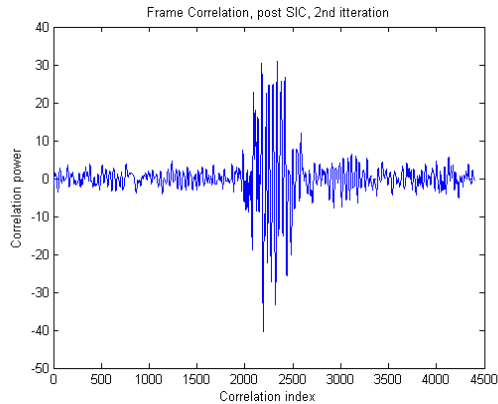


Figure 15 Frame Correlations of SIC Twice

Testing of this demonstration showed that applying SIC a single time to a frame of data would remove, on average, 64 percent of the power of the frame. Applying SIC to the same frame two times would remove, on average, 81 percent of the power of the frame. These averages were generated by testing ten frames of data. Figure 18 shows the cross-correlation of a signal pre and post SIC respectively. It can be seen that the peak, and therefore the power of the signal decreases when SIC is applied to the frame. Figure 19 shows the cross-correlation of the same frame after SIC has been applied twice to the signal. On the second iteration, the demodulation step is skipped. Instead, the previously established remodulated frame is subtracted off of the signal. We can see a further decrease in the correlation from Figure 18 to Figure 19 when SIC is applied.

4.5 Conclusion

Scale Time Offset Robust Modulation (STORM) is a high bandwidth waveform design that adds time-scale to embedded reference modulations using only time-delay. Since STORM can be used as an encapsulation modulation, it can easily be added to current CDMA systems with no additional hardware. There is minimal degradation in conventional performance due to STORM's introduction [40].

Using STORM as a physical layer was proven to offer computational savings for power ordering and synchronization in comparison to a conventional detector. Simulations show that synchronization and power ordering took place at a rate of one thousandth the sample rate. The chosen multiuser detection and mitigation strategy, successive interference cancellation, was shown to be successful in its implementation at removing a significant portion, over sixty percent, of the chosen interfering signals power.

Bibliography

- [1] Shimon Moshavi, "Multi-User Detection for DS-CDMA Communications" IEEE Communications Magazine Vol 34. Issue 10 Oct 1996 pp 124-136
- [2] A. J. Viterbi, "Very Low Rate Convolutional Codes for Maximum Theoretical Performance of Spread-Spectrum Multiple-Access Channels," IEEE JSAC, vol. 8, no 4, May 1990, pp. 641-49.
- [3] R. Kohno et al., "Combination of an Adaptive Array Antenna and a Canceller of Interference for Direct-Sequence Spread-Spectrum Multiple- Access System," IEEE JSAC, vol. 8, no. 4, May 1990, pp. 675-82.
- [4] John G. Proakis "*Digital Communications*". New York, NY: Thomas Casson 2001
- [5] Chen-Chi Hsu & Yumin Lee "A New Multi-Stage Weighted Interference Cancellation Multiuser Detector with User Ordering for DS-CDMA" Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE: Volume 1, 16-20 March 2003 Page(s):595 - 600 vol.1
- [6] Patel P & Holtzman, J. "Analysis of a Simple Interference Cancellation Scheme in a DS-CDMA system". Selected area in Communications, IEEE Journal on : Vol 12, Issue 5 pp 796-807
- [7] Chiasserini, C.F. Rao, R.R." Coexistence mechanisms for interference mitigation in the 2.4-GHz ISM band" Vol 2 Issue 5 pp 964-975
- [8] Thanh, B.N. - Krishnamurthy, V. - Evans, R.J. "Detection aided recursive least squares adaptive MUD in DSCDMA" IEEE Signal Processing Letters Vol9 Issue 8 pp 229-232
- [9] Stefan Franz, "Generalized UWB Transmitted Reference Systems" IEEE Journal on Selected Areas in Communications Vol 24 Issue 4 April 2006 pp 780-786
- [10] Jinho Choi, " Interference mitigation using transmitter filters in CDMA systems" IEEE transactions on Vehicular Technology Vol 51 Issue 4 pp657-666
- [11] Cook, C.; Marsh, H; "An Introduction to Spread Spectrum" IEEE Communications Magazine Vol 21 issue 2 pp 8-16
- [12] Lupas, R.; Verdu, S."Linear Multiuser Detectors for Synchronous Code-Division Multiple-Access Channels" IEEE Transactions on Information Theory Vol 35 Issue 1 pp 123-136

- [13] Varanasi, M.K.; Aazhang, B. "Multistage detection in asynchronous code-division multiple-access communications" IEEE Transaction on Communications Vol 38 Issue 4 1990 pp 509-519
- [14] Zhengyuan Xu; Sadler, B.M. "Multiuser transmitted reference ultra-wideband communication systems" IEEE Journal on Selected Areas in Communications Vol 24 Issue 4 pp 766-772
- [15] Scholtz, R. "The Origins of Spread Spectrum Communications" IEEE Transactions on Communications (legacy – pre 1988) Vol 30 issue 5 part 2 pp 822-854
- [16] Rensheng Wang; Hongbin Li; Tao Li "Robust multiuser detection for multicarrier CDMA systems" IEEE Journal on Selected Area of Communications Vol24 Issue 3 March 2006 pp 673-683
- [17] Pickholtz, R.L.;Milstein, L.B.; Schilling, D.L. "Spread Spectrum for Mobile Communications" IEEE Transactions on Vehicular Technology Vol 40 Issue 2 pp 3123-322
- [18] Pickholtz, R.; Schilling, D.; Milstein, L. "Theory of Spread-Spectrum Communications – A tutorial" IEEE Transactions on Communications (Legacy – pre 1988) Vol 30 Issue 5 Part 2 pp 855-884
- [19] Yi-Ling Chao; Scholtz, R.A "Ultra Wideband Transmitted Reference Systems" IEEE Transactions of Vehicular Technology Vol 54 Issue 5 Sept 2005 pp 1556-1569
- [20] Verdu S. "Minimum Probability of error for Asynchronous Gaussian Multiple Access Channels" IEEE Transactions on Information Theory Vol IT-32 pp 85-96 January
- [21] Verdu S. "Multiple-Access Channels with point-process Observation: Optimum Demodulation" IEEE Transactions on Information Theory Vol IT-32 pp 642-651 September
- [22] Verdu S. "Optimum Multiuser Asymptotic Efficiency" IEEE Transactions on Communications Vol COM-34 pp 890-897 September
- [23] Verdu S. "Recent Progress in Multiuser Detection" *Advances in Communications and Signal Processing*, Springer-Verlag, Berlin. [Reprinted in *Multiple Access Communications*, N. Abramson (ed.), IEEE Press, New York.]
- [24] Verdu S. *Multiuser Detection*, Cambridge University Press, New York.
- [25] Verdu S. "Fifty Years of Communication Theory" IEEE Transactions on Communication Theory, Vol 44 pp 2057-2078 October

- [26] Goldsmith, Andrea "Multiuser Detection" Electrical Engineering 360 – Advanced Topics in Wireless Communications. Lecture 8 – spring 2004. Lecture notes.
- [27] H. Y. Wu and A. Duel-Hallen, "Performance Comparison of Multi-User Detectors with Channel Estimation for Flat Rayleigh Fading CDMA Channels," *Wireless Pers. Commun.*, July/Aug. 1996.
- [28] S. D. Gray, M. Kocic, and D. Brady, "Multi-User Detection in Mismatched Multiple-Access Channels," *IEEE Trans. Commun.*, vol. 43, no. 12, Dec. 1995, pp. 3080-89.
- [29] Simon, Marvin K., Omura, Jim K., Scholtz, Robert A., Levitt, Barry K., "Spread Spectrum Communications Handbook", p. 9, McGraw-Hill, Inc., New York, 1994.
- [30] Batani, N.; "Performance Analysis of Direct Sequence Spread Spectrum with Transmitted Code Reference", *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, Aug 1989, Vol. 2, pp. 731 -735.
- [31] Batani, N.; "Performance Analysis of Direct Sequence Spread Spectrum System with Transmitted Code Reference on Quadrature Carrier", *Proceedings of the 34th Midwest Symposium on Circuits and Systems*, May 1991, Vol.1, pp. 239-244.
- [32] Kennedy, Michael Peter; Kolumbán, Géza; Kis, Gábor; Jákó, Zoltán; "Performance Evaluation of FM-DCSK Modulation in Multipath Environments", *IEEE Transactions on Circuits and Systems*, December 2000, Vol. 47, No. 12, pp. 1702 – 1711
- [33] Young, Randy, K., *Wavelet Theory and its Applications*, Boston, MA: Kluwer Academic Publishers, 1993.
- [34] Hoor, Ralph; Tomlinson, Harold; "Delay Hopped Transmitted Reference RF Communications", 2002 IEEE Conference on UWB Systems and Technologies, pp. 265-269.
- [35] Durgin, G.D.; Kukshya, V.; Rappaport, T.S.; "Wideband Measurements of Angle and Delay Dispersion for Outdoor and Indoor Peer-to-peer Radio Channels at 1920 MHz", *Antennas and Propagation, IEEE Transactions on*, Volume: 51 Issue: 5, May 2003, pp. 936-944.
- [36] Gagliardi, "A Geometrical Study of Transmitted Reference Communication Systems", *IEEE Transactions on Communication Technology*, Volume: 12 Issue: 4, Dec 1964, pp. 118-123
- [37] Proakis, J.G., *Digital Communications*, New York: McGraw-Hill, 1996, pp. 716-717.

[38] Rappaport, T.S., *Wireless Communications*, Upper Saddle River, NJ, 1996, pp.336-338.

[39] Strang *Linear Algebra and Its Applications*, Thomson Higher Education, Belmont, CA 94002, pp160-165

[40] Jenkins, Dave, PhD EE Thesis, "Scale Time Offset Robust Modulation (STORM) for Code Division Multiaccess Multiuser Detection and Mitigation" The Pennsylvania State University, Dec 2008

[41] P. S. Wyckoff; R. K. Young and A. P. Haar; "Scale-Time Offset Robust Modulation (STORM) II Technical Design Fall 2004 Report," *Office of Naval Research Grant No. N00014-04-1-0123*, Jan. 2005

[42] P. Wyckoff ; R. Young; "Comparing STORM and Matched Filtering for Multi-path Channels", *IEEE Consumer Communications and Networking Conference*, pp. 347-351, 5-8 Jan 2004

[43] Wyckoff, P.; Young, R.; McGregor, D; "Scale Time Offset Robust Modulation" *Military Communications Conference, 2003 Milcom 2003 IEEE*, 13-16 Oct 2003. pp 330-335 Vol 1.

[44] Wyckoff, P; Young, R; "STORM Final Report" ONR Grant No. N00014-03-1-0130

Appendix – Matlab File Descriptions

Appendix B MATLAB Files

coherent_detector.m

```
% Alexis Sietins
% 4/2/09
% coherent_detector.m
%
% Function: coherent_detector
%
%Description: This function/file is for debugging purposes. It employes
a
%conventional detector to the given passed data and plots the output of
the
%match filter. This allows a visualization of the timing of the signal.
The
%second part of this file calculates the point in time
%
% Inputs:
%   recv_data - recieved data sequence, the data sequence that will be
fed
%               into the matched filter
%   user_IDs - the other half of the input for the matched filter. The
%               unique data of a particular user
%   search_len - length of the user_IDs that we will correlate with.
%   frame_search_len - amount of adjustment for the search length
%   chip_gain - number of chips per bit
%   frame_size - length, in samples, of a frame of data
%   phase     - length, in samples, between a signal and it's offset
%   itterations - number of times to loop
%   time_scales - time scales to be used in calculating the
offset/scaled
%               signal

function [local_max_index signal_power] = coherent_detector(recv_data,
user_IDs, search_len, ...
    frame_search_len, chip_gain, frame_size, phase, itterations,
time_scales)

%take the number of users off of the user_IDs variable
[num_users cols] = size(user_IDs);

%adjust the search length based on the length of the recieved signal
%if the search_len is short compared to the recv_data, then we will
shorten
%the searcg len to match accordingly
if(search_len > length(recv_data)/2)
    search_len2 = length(recv_data)/2;
else
    search_len2 = search_len + frame_search_len;
end
```



```

%loop
%
% Base loop for the number of users.
for l =1 :num_users;

    %establish the base search code
    search_code2 = user_IDs(l, 1 : frame_size);
    %establish the scaled search code
    search_code3 = resample(search_code2,100*time_scales(1),100);

    %establish the data sequence
    data = recv_data (1:search_len2);
    %    copy_recv_data = recv_data;
    for j = 1:iterations;

        %set the form for both the base code and the scaled code
        data_form = search_code2(1:frame_search_len);
        data_form2 = search_code3(1:frame_search_len);

        %used for debugging purposes
        %    local_max(j) = plot_traditional_correlator( ...
        %        data,data_form);

        %percent error within the data to search
        percent_depth = 3;

        %take the conventional detection (synchronization) of the
signal
        corr_vals = xcorr(data,data_form);
        corr_vals2 = xcorr(data,data_form2);

        %establish the length of the data
        data_len = length(data);

        %establish the length of the data_form
        form_len = length(data_form);

        %
        data_len2 = length(data);
        form_len2 = length(data_form2);

        %IF statement
        %if the received data is longer than the data_form, plot the
desired section
        %of the output of the traditional correlator.
        if(data_len > form_len)
            %The xcorr output a sequence of length 2*M-1, the first
half of this
            %sequence is not desired (not within the timing given),
thus it is
            %tossed out for visual putposes.

```

```

        wanted_corr_vals = abs(corr_vals(data_len:end));
        wanted_corr_vals2 = abs(corr_vals2(data_len:end));

        %figure used for debugging. Plots of the output of the
traditional
        %correlator.
        figure;
        plot(wanted_corr_vals);
        stitle = sprintf('Output of traditional Correlator for user
%i',1);
        title(stitle);
        xlabel('Time (samples)');
        ylabel('correlation value');

        figure;
        plot(wanted_corr_vals2);
        stitle = sprintf('Output for STORM signal for user %i',1);
        title(stitle);
        xlabel('Time (samples)');
        ylabel('correlation value');
    end
    %end IF statement

    %%
    %this section calculates the timing estimate of the
conventional
    %detector. The first local max is taken. There might be more
max
    %local maxes and they might have larger peaks, but the first
one is
    %the only one of real interest.

    %finds the maximum of the absolute value of the
wanted_corr_vals
    max_val = max(wanted_corr_vals);

    %finds all of index's of the values that are close to the
max_val
    index = find(wanted_corr_vals > .9*max_val);

    %sets the error search to look for the actual correlation peak
search_depth = percent_depth*data_len/100;

    % initialize the max value by use of the first value
    local_max = wanted_corr_vals(index(1));
    temp_index = 1;
    %scan through the rest of the want_corr_vals picking out the
max
    %value and storing it
    for k = 2:length(index);
        if(wanted_corr_vals(index(k)) > ...
            wanted_corr_vals(index(temp_index)));
            if(index(k) - index(temp_index) < search_depth)
                temp_index = k;
            end
        end
    end

```

```

        end
    end

    %store the index of the max value of the first peak
    local_max_index(1,j) = index(temp_index);

    %calculate the power of the frame of the signal
    power_signal =
    recv_data(local_max_index(1,j):local_max_index(1,j)+frame_size+phase);
    signal_power(1,j) = sum(power_signal.^2);

    %shift the codes for the next iteration
    search_code2 = circshift(search_code2,[0,-chip_gain]);
end
end

```

```

%Alexis Sietins
%5-14-2009
%
%File: data_collection.m
%
%Description: Record Data from the microphone. Save characteristics
about
%the file too. File needs to be saved manually via the save command at
the
%command line. This is to allow a user specified file name.
%
%Input - none
%
%Output - none

clear all;
close all;

%set the record sampling rate
fs = 44100;

%length of time (in seconds) to record
record_time = 5;

%record the data using the default microphone attached to the computer
data = wavrecord(fs*record_time,fs,1);

%Take the FFT of the recieved data to create the frequency spectrum
fft_data = fft(data);

%calculate the RMS of the recorded
rms_recorded_signal = sqrt(sum(data.^2)/length(data))

%generate a pretty graph of the frequency spectrum
freq = 1/record_time:1/record_time:fs/2;

```

```

figure;
plot(freq(5:end),abs(fft_data(5:length(freq))));
title('frequency spectrum of recorded signal');
xlabel('frequency (Hz)');
ylabel('Energy');

%set user defined specifics about the recorded data. These values will
%be manually changed by the user when the information is needed to be
%changed for testing purposes.
num_users = 1;
% num_phases = 1;
frame_size = 4410;
chip_gain = 441;
time_spread_factors = [.99];
phases = [200];

%Alexis Sietins
%5/17/2009
%
%Decode_data.m
%
%Description: Uses a conventional Coherent correlator to decode the data
%bits given within the passed signal. This function exploits timing
%knowledge of the signal to get a better bit estimate (I.e. it is known
%that each bit should be able the same distance apart from the bits
next to
%it, and therefore the average distance is measured and used to
estimate
%where each bit should be). The recieved data is a STORM code, and the
%conventional detector in this function correlates to not only the
original
%signal, but the scaled and delayed signal as well.
%
%process - this function creates an estimate of what the STORM signal
%should be and correlated to this in the detector.
%
%loop 1: for each bit
%step 1. estimate base signal for the bit
%step 2. copy the signal in step 1.
%step 3. time delay and time scale the copy made in step 2.
%step 4. add signals created in step 1 and step 3.
%step 5. correlate signal in step 4 to the data sequence
%step 6. Store correlator output and the index of the max values of the
% correlator output for each bit
%end loop
%
%step 7. calculate the average bit index and standard deviation of the
bit
% index for each bit. I.e. calculate the average distance between each
% bit.
%step 8. IF (standard deviation is too large) - remove the index's from
% the system that are beyond one standard deviation, thus reducing
the

```

```

% index's standard deviation of values.
%step 9. With the narrowed range of index values created by step 8, do
a
% formal bit detection by taking the sign of the max value of the
reduced
% index range for each bit.
%
%The main loop utilizes a dual sliding window. Both the search_code and
the
%data are moved along the sequence through a sliding window. Both of
these
%windows are moved at the same pace (the chip_gain) on each iteration
of
%the loop. The search_code is comprised of the base signal and the time
%scaled/delayed signal, both of which are moved via a sliding window.
In
%effect, there are three sliding windows being used within the main
loop.
%
%Inputs:
% data - received (recorded) data frame. This sequence should be a
STORM
% signal and it should contain actual data (vs noise). This
sequence
% is longer than it 'needs' to be based off of a timing error
created
% by the uncertainty associated with the timing estimate of the
STORM
% synchronization.
% search_code - data sequence (in this case the user IDs - a ML gold
gold)
% that will be used to correlate against the variable data to
decode
% the information
% chip_gain - chip per bit in the given code. Also known as the
% processing gain of the code.
% time_scale - time scale to be applied to the time delayed part of
the
% STORM Signal - for coherent demodulation.
% delay - timing delay to be applied to the copied signal (before it
is
% time scaled). Used for creating the STORM time-delayed and
scaled
% copy.

% function decoded_bits = Decode_data(data, search_code, chip_gain, ...
% time_scale, delay)
% function decoded_bits = Decode_data(data, search_code, chip_gain, ...
% time_scale, delay)

function [decoded_bits] = decode_frame(S_D, frame_num, user_num)

lower_bound = round(S_D.t_hat(user_num, frame_num) - ...

```

```

        (S_D.search_error_percent/100)*S_D.frame_size);
    if(lower_bound < 1)
        lower_bound = 1;
    end
    upper_bound = round(lower_bound + S_D.frame_size + ...
        2*(S_D.search_error_percent/100)*S_D.frame_size) +
S_D.chip_gain;
    if(upper_bound > length(S_D.recv_data))
        upper_bound=length(S_D.recv_data);
    end

data = S_D.duplicate_recv_data(lower_bound:upper_bound);
search_code = S_D.search_code(user_num,:);
time_scale = S_D.time_spread_factors(user_num);
delay = S_D.phases(frame_num)*S_D.compression_ratio;

%calculates the frame length based off of the length of the search_code
frame_size = length(search_code);

%the number of bits contained within the recieved data signal
num_bits = floor(frame_size/S_D.chip_gain);
%length of the data sequence
data_len = length(data);

%initialize the index vector to 0
index =0;

%transpose the search_code into row vector from colum vector
search_code = search_code';
%scale the search code to desired length
scaled_code = resample(search_code,100*time_scale,100);

%Calculate chips per bit for the scaled signal
scaled_chip_len = S_D.chip_gain*time_scale;

%calculate the scaled delay for the time scaled/delay signal
scaled_delay = delay*time_scale;

%how many samples the signal will be shorted by applying the scale
delay_factor = round(S_D.chip_gain*(1 - time_scale));

%calculate the length of recieved data sequence to use for coherent
%detection. The length is the timing uncertainty (error), plus the
length
%of the delay, plus 1.
corr_len = (data_len - frame_size) + scaled_chip_len +1 + delay;

%initialize and zero out this variable
%This variable will represent the summed original signal and

```

```

%time_scaled/delayed signal to be used in the conventional detector
bit_search_code = zeros([num_bits, delay+S_D.chip_gain]);

%loop
%This loop follows the psuedo code loop discussed in the function
%description section at the beginning of this m file.
for j = 1:num_bits;

    %generate the base signal for conventional detection
    bit_search_code(j, 1:S_D.chip_gain) = search_code(1:S_D.chip_gain);
    %generates time scaled/delayed signal for conventional detection
    bit_search_code(j, scaled_delay-(j-1)*delay_factor: ...
        scaled_delay-j*delay_factor+S_D.chip_gain-1) ...
        = scaled_code(1:S_D.chip_gain-delay_factor);

    %create the output of the conventional detector. Store the data for
    %each bit.
    corr_vals(j, :) = xcorr(data(1 : corr_len), bit_search_code(j, :));

    %following 2 lines are used for debugging. Output of the
conventional
    %correlator based on just the base signal.
    %    simple_corr_vals(j, :) = xcorr(data(1 : corr_len), ...
    %        search_code(1:S_D.chip_gain));
    %
    %these figured used for debugging purposes. Viewing the output of
the
    %correlators via a graph.
    %    figure;
    %    plot(corr_vals(j, :));
    %    figure;
    %    plot(simple_corr_vals(j, :));

    %find and save the index of the max absolute value of the output
    %of the correlator for each bit.
    index(j) = find(max(abs(corr_vals(j, :))) == abs(corr_vals(j, :)));

    %apply the sliding window to the base code
    search_code = circshift(search_code, -S_D.chip_gain);
    %apply the sliding window to the scaled signal
    scaled_code = circshift(scaled_code, -S_D.chip_gain+delay_factor);

    %apply the sliding window to the data
    data = circshift(data, -S_D.chip_gain);

end
%end loop

%%
%this section of the code applies knowledge of the timing of the signal
to
%narrow the search area of the bits. Thus increasing the accuracy of
%correct bit detection

```

```

%take the average of the values in the index vector. I.e. take the
average
%distance between each bit
avg_index = mean(index);
%calculate the statistical standard deviation of the index vector.
std_dev = sqrt(var(index));
%initialize the new_index variable
new_index = [];

%if statement - condition on the standard deviation. Check to see if
the
%standard deviation is small enough to begin with that we don't need to
%reduce it further.
if(std_dev > 2)
    %loop
    %This loop is to reduce the standard deviation to such that we can
    %search for bit timing more accurately.
    for j =1 :num_bits
        %if the measured bit index is outside the standard deviation,
then
        %that is a timing error and that bit index is thrown out in the
        %calculation of the new_index vector.
        if( abs(index(j)-avg_index) < std_dev)
            new_index = [new_index index(j)];
        end
    end
    %end loop
else
    %else, the standard deviation is good enough and we keep the old
values
    new_index = index;
end
%end if statement

%set the maximum and minimum search indexes for bit detection. These
values
%are based off of the narrowed index range of 'new_index' variable.
max_search_val = max(new_index)+1;
min_search_val = min(new_index)-1;

%loop
%This loop does the actual bit detection
for j = 1:num_bits
    %Look at the correlation value of the narrowed range
    true_index(j) = find(max(abs(...
        corr_vals(j, min_search_val:max_search_val)))...
        == abs(corr_vals(j, min_search_val:max_search_val)));

    %formal bit decision
    decoded_bits(j) = sign(corr_vals(j,true_index(j)+min_search_val));

end

% decoded_bits = decoded_bits';

```



```

%these figures are for debugging purposes.
% figure;
% plot(index);
% figure;
% plot(new_index);

% Alexis Sietins
%
%demod_storm_pulse_shape.m
%version 1
% 6/2/2009
%
%
% Description - This m-file is an encapsilation file. As in it controls
the
% entire simulation system just from here. The vfirst section defines
all
% of the parameters that will be used within the simulation system.
%
% step 1. create the unique codes that each user will use to identify
% themselves.
% - main loop -
% step 2 - power order the current frame
% - small loop - for each user
% step 3 - demodulate data
% step 4 - remodulate signal esitmate
% step 5 - subtract
% - end small loop -
%step 6 - advance to next frame
% - end main loop-
%
%
% Over View %
%
clear all;
clear all;
close all;

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BEGIN MAIN %%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% System Variable Delcaration %%%%%%%%%
warning off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% system variables

%load the recorded data file to process on
load 'Z:\MATLAB\recieved data\STORM_signal_5_24_09_rrc_ones_2';
% load 'Z:\MATLAB\recieved
data\STORM_signal_5_21_09_rrc_ones_98scale';
% load 'Z:\MATLAB\recieved data\two_signals_same_synch_2'

```

```

% load 'Z:\\MATLAB\\recieved
data\\STORM_signal_5_21_09_rrc_ones_2users';
% load 'Z:\\MATLAB\\recieved
data\\STORM_signal_5_20_09_rrc_ones_98scale';
% load 'Z:\\MATLAB\\recieved
data\\STORM_signal_5_20_09_rrc_ones_2users';
% load 'Z:\\MATLAB\\recieved data\\STORM_signal_5_20_09_rrc_ones';
% load 'Z:\\MATLAB\\recieved data\\STORM_signal_5_19_09_power_test_2';
% load 'Z:\\MATLAB\\recieved data\\STORM_signal_5_13_09_rrc_ones';
% load 'Z:\\MATLAB\\recieved data\\STORM_signal_5_13_09_rrc_neg_ones';
% load 'Z:\\MATLAB\\recieved data\\STORM_signal_5_12_09_rrc_neg_ones';
% load 'Z:\\MATLAB\\recieved data\\STORM_signal_5_11_09_rrc_ones';
% load 'Z:\\MATLAB\\recieved data\\test_data';

%give string of the filename location containing the desired maximum
length
%gold sequences
S_D.mls_fname = 'Z:\\MATLAB\\PR\\mlseq_13';
%give the string file name location of the filter coefficients used on
the
%recorded data
S_D.filter_fname = 'Z:\\MATLAB\\PR\\Filter_coefs_BP_5_11_09';

% establish some system parameters. Store them in the S_D struct. S_D
% struct contains all revelant System Data to pass from one function to
% another. The next 7 lines of code copy values straight from the data
% file.
S_D.recv_data = data;
S_D.frame_size = frame_size;
S_D.phases = phases;
S_D.num_users = num_users;
S_D.record_time = record_time;
S_D.time_spread_factors = time_spread_factors;
S_D.sample_rate = fs;

%error accountable between the STORM timing estimate and the actual
value
S_D.error = .0025;
% S_D.error = 0;

for j = 1:length(S_D.time_spread_factors)
    S_D.scale_error(j) = (1-S_D.time_spread_factors(j))*100* ...
        S_D.time_spread_factors(j)*S_D.error +
S_D.time_spread_factors(j);
end

% defines the percent to narrow the view of the cross-correlation
output of
% the STORM surface
S_D.percent_view = 20;

%power factor defines how much to multiply the STORM surface peak by to
%achieve the actual signal power. Since there is the base and offset
%signal, the power needs to be multiplied by 2
S_D.power_factor = 2;
%upsampling factor to be applied to the generated code sequence for

```

```

%demodulation and remodulation
S_D.root_raised_cos_factor = 10;
%number of frequency bins to be averaged over. This variable not
currently
%used
S_D.freq_bin_size = 10;
%percent error or the frame between the STORM surface timing estimate
and
%the actual timing value. This is the searchable range on either side
of
%the STORM estimate. As in, the code will search the given percent to
the
%left and to the right of the calculated STORM synch point
S_D.search_error_percent = 2;

%establish the number of users based off the number of spread factors
in
%the given data file
S_D.num_spread_factors = length(S_D.time_spread_factors);

%establish the ratio between the sample rate and frame size.
S_D.compression_ratio = S_D.sample_rate/S_D.frame_size;

%simple boolean expression designating use of the finite impulse
response
%filter on the code sequence for both demodulation and remodulation
S_D.use_filter = true;

%if statement - if (user filter) - then load the filter data into
memory
%and then process the recieved data by the filter.
if(S_D.use_filter)
    %load filter data into memory
    cmd = sprintf('load %s',S_D.filter_fname);
    eval(cmd);

    %same filter_coefs to the system data (S_D) struct
    S_D.filter_coefs = filter_coefs;
    %subtract out the DC term of the recorded code. Put in by the
    %microphone
    S_D.recv_data = S_D.recv_data - mean(S_D.recv_data);

    %filter the recieved code
    S_D.recv_data = filtfilt(...
        S_D.filter_coefs,1,data);
end

%boolean to resume the recieved code to 4410hz
%this code is no longer used
% resample_code = false;

%establish the number of bits each frame of data contains
S_D.bits_per_frame = S_D.frame_size/S_D.chip_gain;

```

```

%upscale the frame_size to match the upsampled rate created by pulse
%shaping the signal - in samples
S_D.frame_size = S_D.frame_size*S_D.compression_ratio;

%upscale the chip_gain in samples to match the upsampled rate created
by
%the pulse shaping of the signal
S_D.chip_gain = S_D.chip_gain*S_D.compression_ratio;

%set a base search length criteria
S_D.search_len = S_D.frame_size;

%establish the number of frames to decode information for
S_D.frames_to_decode = S_D.record_time -1;

% establish the number of divisions that will take place to create the
% STORM surface.
S_D.frame_divisions = 44;
% S_D.frame_divisions = 100;
% S_D.frame_divisions = round(frame_size*(1/time_spread_factors(1) -
1));

%initilize a vartiabile that will contain the information about how much
the
%signal has shifted (I.e. how much the signal has advanced, in samples,
%since the beginning of processing)
total_shift_amount =0;

%clear the data that has been stored in the System Data (S_D) struct
and
%will not be used elsewhere.
clear chip_gain data frame_size num_users
clear record_time time_spread_factors fs fft_data freq phases cmd

%%
%%%%%%%%% Generate User Maximum Length gold code Sequences %%%%%%%%%%%

%generate the user_IDs for each desired user
S_D.base_user_IDs = ...
    generate_user_IDs(S_D.mls_fname, S_D.num_users );

%upsample the user IDs based off of the root raised cosine pulse
shaping
%that took place on the transmitter
S_D.user_IDs = rcosflt(S_D.base_user_IDs, ...
    1,S_D.root_raised_cos_factor,'sqrt');

%transpose the two ID variables so that each row = each user
S_D.user_IDs = S_D.user_IDs';
S_D.base_user_IDs = S_D.base_user_IDs';

```

```

%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MAIN LOOP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initialize the loop for the first frame
current_frame = 1;
%zero out/initialize the matrix containing the decoded bit sequence
S_D.decoded_bits = zeros([S_D.bits_per_frame, S_D.frames_to_decode, ...
    S_D.num_users]);

%establish the section of the user IDs that will be used to create the
%desired frames.
S_D.search_code = S_D.user_IDs(:, 1 : S_D.frame_size);

%make a copy of the recieved signal so that we can shift the copy and
%maintain the reference point with the original
S_D.duplicate_recv_data = S_D.recv_data;

%MAIN LOOP
%
% loops on each frame to be decoded.
% step 1. power order each user - obtain timing estimate at same time
% step 2. demodulate user with highest power
% step 3. remodulate user with highest power
% step 4. subtract out signal from total
% goto step 2
%
%advance loop
while(current_frame <S_D.frames_to_decode+1)

    %power oder each user fro this frame with the power_order function
    [S_D.power_estimate(:,current_frame),S_D.t_hat(:,current_frame)]
=...
        power_order(S_D);

    %loop - over each user
    %1. obtain user with highest power
    %2. demodulate user.
    %3. remodulate user
    %4. subtract out user from signal
    for j =1:S_D.num_users;

        %establish the user with the highest power.
        user_to_be_removed = ...
            find(max(S_D.power_estimate(:,current_frame)) == ...
                S_D.power_estimate(:,current_frame));

        %decode data for user with highest power.
        S_D.decoded_bits(:,current_frame, user_to_be_removed) =...
            decode_frame(S_D,current_frame,user_to_be_removed);

        %using data decoded from previous step, remodulate the signal
    for

        %the user with the highest power
        [S_D.remoddodded_signal] = ...

```

```

        remod_signal(S_D,user_to_be_removed, current_frame);

        %subtract out the signal from the agrigate signal
        S_D.duplicate_recv_data = ...
            remove_frame_from_signal(S_D, user_to_be_removed,
current_frame);

        %debugging purposes.
        %figure plots the power, based off of corrlation, of each user.
%       figure;
%       plot(xcorr(S_D.recv_data,S_D.remodded_signal));
%       title('correlation of old signal');

        %zero out the user with the highest power so that the next user
can be
        %removed
        S_D.power_estimate(user_to_be_removed, current_frame) = 0;

    end
    %END USER LOOP

    %advance the current frame
    current_frame = current_frame+1;
%    current_frame = current_frame+6;

    %shift the recv_data vector to the next frame.
    S_D.duplicate_recv_data = ...
        circshift(S_D.duplicate_recv_data ,-1.3*S_D.frame_size);

    %add up the shifted amount and keep track of how much has been
shifted.
    total_shift_amount = total_shift_amount + S_D.frame_size;
end

%END MAIN LOOP

%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Perform Traditional Correlation %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% I.e. matched filter

%this section is purley for debugging purposes.
%applied a matched filter for a timing estimate to the first frame of
%recorded data
if(false)
    frame_search_len = 30000;
    itterations = 1;
    [S_D.coherent_synch_pt S_D.signal_power] = ...
        coherent_detector(S_D.recv_data, S_D.user_IDs, S_D.search_len,
...
        frame_search_len, S_D.chip_gain, S_D.frame_size, ...
        S_D.phases, itterations, S_D.time_spread_factors);

```

end

```
%Alexis Sietins
%5/14/2009
%
% Function: find_sync.m
%
%Description: Take a STORM surface matrix and calculates a timing
%estimate. First, a 2-d projection of the storm surface is created.
% Then the largest (longest) peak is taken to lock onto. As assumption
is
% being made that the longest storm surface will belong to the storm
% signal that is most in view of the given frame. Next, using a 2nd
order
% linear least squares form fitting function, the equation of the line
% associated with the chosen STORM surface is estimated. Combining the
% knowledge of the line and the given delay estimate, a timing estimate
% is created and returned
%
% inputs:
%     input - STORM surface. N x M sized matrix. Rows - Time divisions.
%           Corrasponds to time. Colums - correlation output. Corresponds
%           to delay. N << M normally
%     tau - delay estimate to be used in combination with STORM surface
line
%           equation to generate timing estimate.
% Output:
%     pt - timing estimate
```

```
function [pt, power_estimate] = find_sync(input,tau)
```

```
%fins the number of rows and columns of input - the STORM surface
[rows cols] = size(input);
```

```
%Pre-loop
```

```
%initialize loop conditions by setting the first peak of the storm
surface
```

```
peak(1) = find(max(input(1,:)) == input(1,:));
```

```
%intiaailize loop
```

```
frame_bgn_index = [1];
```

```
%Loop
```

```
%generate 2-D STORM surface projection
```

```
for j = 2:rows
```

```
    peak(j) = find(max(input(j,:)) == input(j,:));
```

```
    power(j) = max(input(j,:));
```

```
    if(peak(j) > peak(j-1))
```

```
        frame_bgn_index = [frame_bgn_index , j];
```

```
    end
```

```

end

frame_bgn_index = [frame_bgn_index , rows];

%Loop
%calculate the length of each surface within the given Frame
for j = 2:length(frame_bgn_index)
    lens(j-1) = frame_bgn_index(j)-frame_bgn_index(j-1);
end

% while(pt<0)
%find the frame with the most data points associated with it
full_frame = find(max(lens) == lens);
if(size(full_frame) >1)
    full_frame = full_frame(1);
end

%create the power estimate based of the max value on the desired
%correlation range
power_estimate = max(power(frame_bgn_index(full_frame):...
    frame_bgn_index(full_frame+1)));

%%
%form fitting
%2nd order Linear Least Squares Form Fitting
%
% Typical least squares solution
% [b0 b1 ] = inv(X'*X)*X'*Y
% where
% X - each column of X is the x component of the data
% vector raised to the power of the
% column number (Ex. 1st col is all 1's. 2nd col is actual raw
data. 3rd
% col is data squared... ect)
% Y - y component of the data vector M x 1 vector
%
% This system obtimizes  $y = b_0 + b_1*x$ 

%generate X matrix
% X = [ones(1,lens(full_frame)) ; ...
%
peak(frame_bgn_index(full_frame):frame_bgn_index(full_frame+1)-1]);
X = [ones(1,lens(full_frame)) ; ...
    frame_bgn_index(full_frame):frame_bgn_index(full_frame+1)-1];
X = X';
%generate Y vector
% Y = frame_bgn_index(full_frame):frame_bgn_index(full_frame+1)-1;
Y = peak(frame_bgn_index(full_frame):frame_bgn_index(full_frame+1)-
1);
Y = Y';

%Solve for B vector
B_hat = inv(X'*X)*X'*Y;

```



```

figure;
plot(peak);
title('2-d projection of STORM surface');
xlabel('Correlation index (time)');
ylabel('time delay (samples)');

%adjust the giving delay estimate for the middle of the correlation
tau_hat = (length(input(1,:))/2) + tau;
%use B vector solution to create timing estimate
%   pt = B_hat(1) + B_hat(2)*tau_hat;
pt = (tau_hat-B_hat(1))/B_hat(2);

```

```

% Alexis Sietins
% 4/3/2009
% generate_user_IDs.m
%
% Description: This file creates the maximum length gold sequences for
each
% user. The number of users is passed as an argument into the function.
% Since generating these values cold each time can take a long time, the
% maximum length base sequences needed to create the gold code family
have
% previously been calculated and saved to the hard disk. These values
are
% loaded into memory and then xor'd to obtain the unique id for each
user.
%
% Input -
%   fname - string file name used to locate the maximum length sequence
%           generating codes
%   num_users - number of users. Each user will get a unique code
assigned
%           to them
%
% Output -
%   user_IDs - [n x m] matrix containing the codes for each user. n is
the
%   number of users, as designated by the num_users input variable. m is
the
%   2^(number of shift registers)-1 - where (number of shift registers)
is
%   deturmined by the fname passed to the function

```

```
function [user_IDs ] = generate_user_IDs(fname, num_users )
```

```
%%%%%%%%%%%% generate User IDs %%%%%%%%%%
```

```

%store the filename into a command line string and then execute the
%command such that the file is laoded into local memory

```

```

cmd = sprintf('load %s',fname);
eval(cmd);
% load 'C:\\Documents and Settings\\Administrator\\My
Documents\\MATLAB\\PR\\mlseq_13'
%     load 'Z:\\MATLAB\\PR\\mlseq_13';

%compare the lengths of the two sequences. If they are not the same
%length, then there is a problem and exit
if(length(ml_seq1) ~= length(ml_seq2))
    error('Maximal length sequences are not' ...
        'of equal length - exiting now');
    quit;
else
    %if they are the same length, establish the baseline length for
the
    %system.
    mls_len = length(ml_seq1);
end

%%%%%%%%%% Set up information needed for each user %%%%%%%%%%%

%initilize the variable user_IDs
user_IDs = zeros(num_users,mls_len);

%convert the ML sequences to binary 0 or 2
%the xor function needs 0 and non-zero entries
ml_seq1 = ml_seq1 + 1;
ml_seq2 = ml_seq2 + 1;

%loop spans the GOLD Space defined by the ML sequences.
%each row is the ID for each specific user
for j = 1:num_users;
    %take the xor of the 2 sequences to create each user id
    user_IDs(j,:) = xor(ml_seq1,ml_seq2);

    %do a logical circular shift on 1 register.
    temp = ml_seq1(1);
    ml_seq1(1:mls_len-1) = ml_seq1(2:mls_len);
    ml_seq1(mls_len) = temp;
end

%convert user_IDs from logical 0 & 1 to logical -1 & 1
user_IDs = 2.*user_IDs - 1;
%make a copy. 1 to be used for transmitting, the other recieving
user_IDs = user_IDs';

%Alexis Sietins
% 4/3/2009
% power_order.m
%
```

```

%
%Description: Create a power ordered estimate for each user.
%      step 1; creat a storm surface for each user
%      step 2; calcualte a timing estimate and power estimate based
off of
%      the storm surface for each user
%
%Input
%  S_D - System Data Struct. Contains all relevant system data
%
% Output
%  power_estimate - vector containing the power estimates for each user
%  timing_estimate - vector containing all the STORM timing estimates
%

function [power_estimate, timing_estimate] = power_order(S_D)

%expand the searchable range - since the MATLAB transmit capabilities
are
%not real time and therefore have a gap between each transmitted frame
search_len = 1.3*S_D.frame_size;

%amount, in samples, the frame should be shifted each itteration
shift_size = round(search_len/S_D.frame_divisions);

%eastblish a copy of the recv_data to shift
duplicate_data = S_D.recv_data;

%outter loop
% for each user
%1. create STORM surface
%2. take power estimate (max of the storm surface)
%3. create timing estimate based off of the STORM surface
for j = 1:S_D.num_users

    %grab the data to be turned into a STORM surface
    shifted_recv_data = duplicate_data(1:2*search_len)';

    %loop - for each frame division
    % this loop creates the STORM surface
    for k = 1:S_D.frame_divisions

        %establishes the base signal
        unscaled_recv_data = shifted_recv_data(1:S_D.frame_size);
        %establishes the scaled signal
        scaled_recv_data = resample(unscaled_recv_data , ...
            100*S_D.time_spread_factors(j), 100);

        %autoabiguity function for the scaled & base signal
        corr_vals(j,k,:) = xcorr(unscaled_recv_data', ...
            scaled_recv_data', 'none');

        %time shift the data sequence
        shifted_recv_data = ...
            circshift(shifted_recv_data, [0,-shift_size]);
    end
end

```

```

end

%establish the length of the cross correlation output
corr_len = length(corr_vals(j,1,:));

%next few lines compress the "view" of the cross-correlation such
that
%it is easier to view and easier to process.
num_samples_view = (S_D.percent_view/100)*corr_len;
upper_bnd = round((corr_len/2) + (num_samples_view/2));
lower_bnd = round((corr_len/2) - (num_samples_view/2));
%store the 'narrowed view' signal in a temp variable
temp(:, :) = corr_vals(j, :, lower_bnd:upper_bnd);

%establish the delay value that will be used to obtain the timing
%estimate
real_tau(j) = round(S_D.phases(j)*S_D.time_spread_factors(j))*...
    S_D.compression_ratio;

%obtain the timing estimate. This value is in
%frame/frame_divisions
[pt_hat(j), power_estimate(j)] = ...
    find_sync(temp, real_tau(j)*S_D.scale_error(j));

%convert the timing estimate created by the find_sync function into
the
%desired sample
timing_estimate(j) = pt_hat(j)*search_len/S_D.frame_divisions - ...
    real_tau;

end

```

```

%Alexis Sietins
% remod_signal.m
% 5/14/2009
%
% Function : remod_signal
% Description: create an estimate of a transmitted signal to be
% subtracted out from a recieved signal for Sucessive
% Interference Cancellation (SIC)
% The output has an unmodified power spectrum - I.e. the signal
% needs to be further modified to account for recieved power.
%
% Inputs:
% bit_sequence - estimated recieved bits
% user_ID - gold code sequency comprising unique user IDs
% chip_gain - number of chips per bit
% delay - time delay shift between the base signal and the
scaled/delayed

```

```

%         signal
%     time_spread_factor - scaling factor to be used on the
scaled/delayed
%         signal
%     filter_coefs - filter coefficients to be used after pulse shaping
% Outputs:
%     rx - estimated recieved signal, with no adjustment for power

function [rx] = remod_signal(S_D, user_num, frame_num)

bit_sequence = S_D.decoded_bits(:,frame_num);
chip_gain = S_D.chip_gain/S_D.compression_ratio;
user_ID = S_D.base_user_IDs(user_num,:);
delay = S_D.phases(frame_num);
time_spread_factor = S_D.time_spread_factors(user_num);
filter_coefs = S_D.filter_coefs;
power_estimate = S_D.power_estimate(user_num, frame_num);

%factor to be used with the root raised cosine filter. In other
%words, the amount of upsampling that takes place
root_raised_cos_factor = 10;

%temp variable so it can be circular shifted without loosing the
%original begining
user_ID_copy = user_ID;

%number of decoded bits
num_bits = length(bit_sequence);

%length of the signal
signal_len = num_bits*chip_gain;

%if statement checking the the user_id is of sufficient length
%if the ID is not long enough, then there will be redudant correlation
%within a single data Frame
if(signal_len > length(user_ID))
    'error - remod_signal.m User ID sequence not long enough'
    rx =0;
    return
end

%initialize the base signal and zero it out
base_signal = zeros([1,signal_len]);

%loop: generates the base signal.
for j = 1:num_bits
    %multiplies segments of the user_ID by the estimated bit sequence
    base_signal((j-1)*chip_gain+1:j*chip_gain) = ...
        bit_sequence(j)*user_ID_copy(1:chip_gain);
end

```

```

    %rotates user_ID_copy by the chip gain. This rotation is needed for
    %the previous line of code
    user_ID_copy = circshift(user_ID_copy,[0,-chip_gain]);
end

if(S_D.bits_per_frame - floor(S_D.bits_per_frame) > 0)
    additional_sig_len = S_D.frame_size/S_D.compression_ratio -
    signal_len;
    base_signal = [base_signal user_ID_copy(1:additional_sig_len)];
end

%add a tail of zeros to the base signal - so that when base signal is
%copied and delayed, the base signal won't rotate around on itself.
offset_signal = [base_signal [zeros([1,delay])]];
%copy and time delay the base signal
offset_signal = circshift(offset_signal, [0,delay]);
%time-scaled the delayed/copied signal by the time_scaling factor
scaled_offset_signal = resample(offset_signal,...
    100*time_spread_factor, 100);

%calculate the number of samples to extend the scaled/offset signal
% pad_len = length(base_signal) - length(scaled_offset_signal);

%pad the end of the scaled/delayed signal so its length matched the
length
%of the base signal
% scaled_offset_signal = scaled_offset_signal';
% scaled_offset_signal = [scaled_offset_signal ; zeros([pad_len,1])];
% scaled_offset_signal = scaled_offset_signal';

%pulse shape the base signal
pulse_shaped_base =
rcosflt(base_signal,1,root_raised_cos_factor,'sqrt');
%filter the pulse-shaped base signal
filtered_base_sig = filtfilt( filter_coefs, 1, pulse_shaped_base);
% filtered_base_sig = filtfilt( filter_coefs, 1, filtered_base_sig);
% final_base_sig =
rcosflt(first_filtered_base_sig,1,root_raised_cos_factor,'sqrt');

%pulse shape the time sclaed/delayed signal
pulse_shaped_scaled_sig = rcosflt(...
    scaled_offset_signal,1,root_raised_cos_factor,'sqrt');
%filter the pulse-shaped time scaled/delayed signal
filtered_scaled_sig = filtfilt( filter_coefs, 1,
pulse_shaped_scaled_sig);
% filtered_scaled_sig = filtfilt( filter_coefs, 1,
filtered_scaled_sig);

base_power = sum(filtered_base_sig.^2);
scaled_power = sum(filtered_scaled_sig.^2);
%
% filtered_base_sig2 =
filtered_base_sig*(sqrt(power_estimate/base_power));

```

```

% filtered_scaled_sig2 =
filtered_scaled_sig*(sqrt(power_estimate/scaled_power));
%
filtered_base_sig =
filtered_base_sig*(sqrt(power_estimate/base_power));
filtered_scaled_sig =
filtered_scaled_sig*(sqrt(power_estimate/scaled_power))*time_spread_factor;

% base_power2 = sum(filtered_base_sig2.^2)
% scaled_power2 = sum(filtered_scaled_sig2.^2)

%Add the two signals together for the final signal estimate
rx = zeros([length(filtered_scaled_sig),1]);
rx(1:length(filtered_base_sig)) =rx(1:length(filtered_base_sig))+
filtered_base_sig;
rx(1:length(filtered_scaled_sig)) = rx(1:length(filtered_scaled_sig)) +
filtered_scaled_sig;
rx(1:length(filtered_scaled_sig))=
filtfilt(S_D.filter_coefs,1,rx(1:length(filtered_scaled_sig)));
% rx(1:length(filtered_scaled_sig)) =
rx(1:length(filtered_scaled_sig))*2*sqrt(power_estimate);

base_power = sum(filtered_base_sig.^2)
scaled_power = sum(filtered_scaled_sig.^2)
total_power = sum(rx.^2)
%
% rx2 = zeros([length(filtered_scaled_sig2),1]);
% rx2(1:length(filtered_base_sig2)) =rx2(1:length(filtered_base_sig2))+
filtered_base_sig2;
% rx2(1:length(filtered_scaled_sig2)) =
rx2(1:length(filtered_scaled_sig2))+ filtered_scaled_sig2;

%Alexis Sietins
%4/3/2009
%remove_frame_from_signal.m
%
% description: Apply the subtraction of the remodded signal to the
recorded
% signal. First, do a fine accurate synchronization to the remodded
frame.
% Then subtract out the signal based on this fine timing estimate.
%
% Input -
%   S_D - struct containing all of the system data
%   user_num - the number of the current user being processed.
%   frame_num - number of the current frame being processed
%
% Output -
%   rx_signal - new signal with the frame subtracted from it

```

```

function rx_signal = remove_frame_from_signal(S_D, user_num, frame_num)

%establish the length of the remodded signal
remod_len = length(S_D.remodded_signal);

%establish the data that will be searched for the fine synchronization
searchable_data = S_D.duplicate_recv_data(1:2*S_D.frame_size);

%calculate the cross correlation
corr_vals = xcorr(searchable_data,S_D.remodded_signal);

%establish the point in the corr_vals variable that the search will
start
%with
min_search_index = 2*S_D.frame_size;

%calcuate the fine tune timing estimate
timing_estimate = find(max(abs(corr_vals(min_search_index:end))) == ...
    abs(corr_vals(min_search_index:end)));
sgn = sign(corr_vals(min_search_index+timing_estimate));

%these figures are used for debugging purposes.
% figure;
% plot(corr_vals);
% figure;
% plot(corr_vals(min_search_index:end));

%estimate the power based off the maximum corrleation value
power_est = max(corr_vals(min_search_index:end));
%calculate the power of the remodded signal
remod_power = sum(S_D.remodded_signal.^2);

%scale the remodded signal based on the power estimate
remod_sig = S_D.remodded_signal*power_est/remod_power;

%copy the duplicate_signal so that subtraction can take place
rx_signal = S_D.duplicate_recv_data;

%begugging purposes
%estimate tthe power of the frame before subtraction
% base_rd_power_est = sum(...
%     rx_signal(timing_estimate:timing_estimate + remod_len).^2)

%SUBTRACT out the reomded signal from the signal
rx_signal(timing_estimate:timing_estimate + remod_len -1) = ...
    rx_signal(timing_estimate:timing_estimate + remod_len -1) - ...
    sgn*remod_sig;

%debugging purpose
%calcualte the power of the new signal
% new_rx_power_est = sum(rx_signal(timing_estimate:timing_estimate...
%     + remod_len).^2)
% remod_power_est = sum(S_D.remodded_signal.^2)

```



```

%debugging purpose
% figure plots the correlation peaks of the frame with the demodded
signal
% figure;
% plot(xcorr(rx_signal,S_D.remodded_signal));
% title('correlation of new signal');

% Alexis Sietins
%transmit_storm_pulse_shape.m
% 4/4/2009

%Description: Transmit an acoustic STORM signal
%conditions wanted in this file:
% 1. supports multiple users
% 2. create a signal for each user,
% 3. each user has a specified time scale and time delay
% 4. each user can have an adjustable power
% 5. each user can have adjustable delay (timing erro) introduced into
the
% signal
% 6. play the composit signal for a certain length of time, repeating
the
% signal

clear all;
clear all;
close all;

%%
%%%%%%%%%% system variables %%%%%%%%%%%

%load the file containing the filter coefficients used to sven out the
%spectrum for transmitssion
load 'C:\\Documents and Settings\\Administrator\\My Documents'...
'\\MATLAB\\PR\\Filter_coefs_BP_5_11_09.mat'

num_users = 1; %number of users

chip_gain = 441; %number of chips per bit.

NSR = 13; %number of shift registers used in calculating the MLS
sequence

```

```

sample_rate = 44100;
root_raised_cos_factor = 10;

play_time = 10;
frame_size = 4410;
time_spread_factor = [.99];
num_samples = frame_size;
phase_delay = [200];
% num_samples = 5*sample_rate;
frame_num = 1;

compression_ratio = sample_rate/frame_size;

if(2^(NSR)-1 < num_users)
    error('MLS sequence too short for number of users - exiting now');
    quit;
end

if(2^(NSR)-1 < frame_size)
    error('MLS sequence too short for frame size - exiting now');
    quit;
end

is_running = false;
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Generate ML gold sequences %%%%%%%%%%%%%%%

%load the file containing the maximum length gold codes. These
sequences
%have been calculated apriori to reduce run-time computation time.
load 'C:\\Documents and Settings\\Administrator\\'...
    'My Documents\\MATLAB\\PR\\mlseq_13';

%if statement
%establish that the two sequences are of the same length
if(length(ml_seq1) ~= length(ml_seq2))
    %if the two sequences are not of the same length, we cannot
    continue.
    error('Maximal length sequences are not of equal'...
        ' length - exiting now');
    quit;
else
    %establish the length of the sequences, used in the rest of the
    %program. This length will be 2^(number of shift registers)-1,
    where
    %(number of shift registers) is a variable used in calculating the
    %maximum length sequences
    mls_len = length(ml_seq1);
end

%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set up information needed for each user %%%%%%%%%%%%%%%

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Generate Needed Psuedo Random Sequences

% creates 2-D matrix to be filled with the needed unique user ID's
% each row represents a seperate user
user_IDs = zeros(num_users,mls_len);

%convert the ML sequences to binary 0 or 2
%the xor function needs 0 and non-zero entries
ml_seq1 = ml_seq1 + 1;
ml_seq2 = ml_seq2 + 1;

%loop spans the GOLD Space defined by the ML sequences.
%each row is the ID for each specific user
for i = 1:num_users;
    %take the xor of the 2 sequences to create each user id
    user_IDs(i,:) = xor(ml_seq1,ml_seq2);

    %do a logical circular shift on 1 register.
    temp = ml_seq1(1);
    ml_seq1(1:mls_len-1) = ml_seq1(2:mls_len);
    ml_seq1(mls_len) = temp;
end

%convert user_IDs from logical 0 & 1 to logical -1 & 1
user_IDs = 2.*user_IDs - 1;
%make a copy. 1 to be used for transmitting, the other recieving
user_IDs_recv = user_IDs;

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Create Transmitted signal

%zero out/initialize the transmitted signal
transmitted_sig = zeros(1,compression_ratio*length(user_IDs(1,:)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Generate Bit sequences (simulated transmitted Data) at
Random

%created large 2-D matrix with random binary data.
%user_data_sequence = -1*ones(num_users, round(num_samples/chip_gain));
user_data_sequence = ones(num_users, round(num_samples/chip_gain));

%zero out these zariables for future use
current_max_len = 0;
previous_max_len = 0;

%loop
%outter loop to shift through each user
for j = 1:num_users;

    %loop - shift through each bit
    % create the base signal for the first user
    for k = 1:num_samples/chip_gain

```

```

        %create a temp variable - will make shifting the User_ID much
        %easier.
        temp = user_IDs(j,1:chip_gain);
        %signal_idx used to visually simplify the update equation
        sig_index = (k-1)*chip_gain + 1 ;

        %update the transmitted signal with the new chip pattern for
the
        %given user, user_ID, and bit value
        base_sig(sig_index: sig_index + chip_gain - 1) = ...
            user_data_sequence(j,k)*temp;

        %circular shift the MLS sequences
        user_IDs(j,1:mls_len-chip_gain) =
user_IDs(j,chip_gain+1:mls_len);
        user_IDs(j,mls_len - chip_gain + 1:mls_len) = temp;
    end

    %expand the base signal so that is can be circular shifted without
    %loosing data
    expanded_base_sig = [base_sig zeros([1,phase_delay(j)])];

    %circular shift (delay) the base signal to create the time delayed
copy
    %of the base signal
    delayed_base_sig = circshift(expanded_base_sig, [0,
phase_delay(j)]);

    %scale the time delayed base signal to create the time
delayed/scaled
    %signal
    scaled_sig = resample(delayed_base_sig,100*time_spread_factor(j),
100);

    %pulse shape the base signal
    rrc_base_sig = rcosflt(base_sig,1, root_raised_cos_factor,'sqrt');
    % filter the pulse shaped signal for spectral reasons
    filtered_base_sig = filtfilt(filter_coefs,1,rrc_base_sig);

    %pulse shape the time delayed/scaled signal
    rrc_scaled_sig = rcosflt(scaled_sig,1,
root_raised_cos_factor,'sqrt');
    %filter the pulse shpaed scaled signal for spectral reasons
    filtered_scaled_sig = filtfilt(filter_coefs,1,rrc_scaled_sig);

%    have an adjustable pad length for debugging purposes.
    pad_len = 0;

    %add the base signal to the transmitted_sig
    transmitted_sig(1+pad_len:length(filtered_base_sig)+pad_len) =...
        transmitted_sig(1:length(filtered_base_sig)) +
filtered_base_sig';

    %add the scaled/delayed signal the to transmitted_sig

```

```

transmitted_sig(1+pad_len:length(filtered_scaled_sig)+pad_len) =...
    transmitted_sig(1:length(filtered_scaled_sig)) +...
    filtered_scaled_sig';

%establish what signal has the greater length.
current_max_len = max([length(filtered_base_sig) ...
    length(filtered_scaled_sig)]);

%compare the max length of the current signal to previous signals.
If
    %the new one is longer, establish the new length to be the
    greatest.
    if(current_max_len > previous_max_len)
        previous_max_len = current_max_len;
    end

end

%establish the signal that will actually be transmitted
final(1,1:previous_max_len) = transmitted_sig(1:previous_max_len);
%only one speaker should play, this the second channel should be zeroed
final(2,1:previous_max_len) = zeros([1,previous_max_len]);

%play the desired signal until the play time has been exceeded
tic
while(toc<play_time)
    wavplay(final',sample_rate);
end

```